Petri Net

Theory and Applications

Edited by
Vedran Kordic

I-TECH Education and Publishing

Published by the I-Tech Education and Publishing, Vienna, Austria

www.i-techonline.com
Additional copies can be obtained from:
publication@i-techonline.com

First published February 2008
Printed in Croatia

A catalog record for this book is available from the Austrian Library.
Petri Net, Theory and Applications, Edited by Vedran Kordic
p. cm.
ISBN 978-3-902613-12-7
1. Petri Net. 2. Theory. 3. Applications.

Preface

Although many other models of concurrent and distributed systems have been developed since the introduction in 1964 Petri nets are still an essential model for concurrent systems with respect to both the theory and the applications.

The main attraction of Petri nets is the way in which the basic aspects of concurrent systems are captured both conceptually and mathematically. The intuitively appealing graphical notation makes Petri nets the model of choice in many applications. The natural way in which Petri nets allow one to formally capture many of the basic notions and issues of concurrent systems has contributed greatly to the development of a rich theory of concurrent systems based on Petri nets.

This book brings together reputable researchers from all over the world in order to provide a comprehensive coverage of advanced and modern topics not yet reflected by other books. The book consists of 23 chapters written by 53 authors from 12 different countries.

In the name of I-Tech, editor is very much indebted to all the authors entrusted us with their newest research results.

Contents

1

Petri Net Transformations

Hartmut Ehrig, Kathrin Hoffmann, Julia Padberg, Claudia Ermel,
Ulrike Prange, Enrico Biermann and Tony Modica
Institute for Software Technology and Theoretical Computer Science
Technical University of Berlin
Germany

1. Introduction

Modelling the adaption of a system to a changing environment gets more and more important. Application areas cover e.g. computer supported cooperative work, multi agent systems, dynamic process mining or mobile networks. One approach to combine formal modelling of dynamic systems and controlled model adaption are Petri net transformations. The main idea behind net transformation is the stepwise development of place/transition nets by given rules. Think of these rules as replacement systems where the left-hand side is replaced by the right-hand side while preserving a context. This approach increases the expressiveness of Petri nets and allows in addition to the well known token game a formal description of structural changes.
The chapter is structured as follows: We start with a general overview of net transformations [25, 30, 7, 10] in Section 2. In Section 3, we illustrate the rule-based refinement of place/transition nets in terms of a case study in the area of an emergency scenario [4]. The case study shows how to use Petri net transformations as refinement concept and demonstrates the compatibility of net refinement and net composition which indicate the relevance of Petri net transformations for software engineering. In Section 4, we present precise definitions of basic notions concerning Petri net transformations in the case of place/transition nets. The union theorem shows the compatibility of net transformations with the union of nets via a common interface provided that the net transformations are preserving this interface. Furthermore, results for high-level nets are also briefly discussed at the end of Section 4. In the conclusion we discuss how tools for graph transformation systems can also be used for Petri net transformations.

2. General overview of net transformations

The main idea of net transformations is the rule-based modification of nets where each application of a rule leads to a net transformation step. While the well-known token game of Petri nets does not change the net structure, the concept of Petri net transformations is a rule-based approach for dynamic changes of the net structure of Petri nets. Since Petri nets can be considered as bipartite graphs the concept of graph transformations can be applied to define transformations of Petri nets. In the following we give a general overview of graph and net transformations, for more details see [30, 8, 12, 7, 14].
The research area of graph transformation is a discipline of computer science which dates back to the early seventies. Methods, techniques, and results from the area of graph

transformation have already been studied and applied in many fields of computer science such as formal language theory, pattern recognition and generation, compiler construction, software engineering, concurrent and distributed systems modelling, database design and theory, logical and functional programming, AI, visual modelling, etc. Graph transformation has at least three different roots, namely from Chomsky grammars on strings to graph grammars, from term rewriting to graph rewriting, and from textual description to visual modelling.

Computing by graph transformation is a fundamental concept for programming, specification, concurrency, distribution, and visual modelling. A state of the art report for applications, languages and tools for graph transformation on the one hand and for concurrency, parallelism and distribution on the other hand is given in volumes 2 and 3 of the *Handbook of Graph Grammars and Computing by Graph Transformation* [8] and [12]. In our paper [14], we have presented a comprehensive presentation of graph and net transformations and their relation. Petri net transformations can also be realized for algebraic high-level nets [25], which is a high-level net concept integrating algebraic specifications with place/transition nets.

In contrast to most applications of the graph transformation approach, where graphs denote states of a system, and rules and transformations describe state changes and the dynamic behavior of systems, in the area of Petri nets we use rules and hence transformations to represent stepwise modification of nets. This kind of transformation for Petri nets is considered to be a vertical structuring technique, known as rule-based net transformation. Basically, a rule (or production) $r = (L, R)$ is a pair of graphs (or nets) called left-hand side L and right-hand side R. Applying the rule $r = (L, R)$ means to find a match of L in the source graph (or net) and to replace L by R. In order to replace L by R we need to connect R with the context leading to the target graph (respectively the target net) of the transformation.

The well-known argument in favour of formal techniques, to have precise notions and rigid mathematical results, clearly holds for this approach as well. Moreover, we have already investigated net transformations in high-level Petri net classes (see Subsection 4.6) that are even more suitable for system modelling than the place/transition nets in our case study. The impact for system development is founded in what results from net transformations:

- *Stepwise Development of Models:* The model of a complex software system may reach a size that is difficult to handle and may compromise the advantages of the (formal) model severely. The one main counter measure is breaking down the model into sub-models, the other is to develop the model top-down. In top-down development the first model is a very abstract view of the system and step by step more modelling details and functionality are added. In general, however, this results in a chain of models that are strongly related by their intuitive meaning, but not on a formal basis. Petri net transformations fill this gap by supporting the formal step-by-step development of a model. Rules describe the required changes of a model and their applications yield the transformations of the model. Moreover, the representation of changes in a visual way using rules and transformations is very intuitive and does not require a deeper knowledge of the theory.
- *Distributed Development of Models:* Decomposing a large model is an important technique for the development of complex models. To combine the advantages of a horizontal structuring with the advantages of step-by-step development, vertical structuring techniques for ensuring the consistency of the composed model are required. Then a distributed step-by-step development is available that allows the independent development of submodels. The theory of net transformation comprises horizontal

structuring techniques and ensures compatibility between these and the transformations. In Subsection 4.4 we introduce the union construction for the decomposition, and the union theorem in Subsection 4.5 allows to develop the subnets independently of each other. The theory allows complex compositions and decompositions, where the independence of the sub-models is essential. So, the formal foundation for the distributed development of complex models is given.

- *Incremental Verification:* Pure modification of Petri nets is often not sufficient, since the net has some desired properties that have to be ensured during further development. Verification of each intermediate model requires a lot of effort and hence is cost intensive. But refinement can be considered as the modification of nets preserving desired properties. Hence the verification of properties is only required for the net where they can be first expressed. In this way properties are introduced into the development process and are preserved from then on. Rule-based refinement modifies Petri nets using rules and transformations so that specific system properties are preserved. For a brief discussion see Subsection 4.6.
- *Foundation for Tool Support:* A further advantage is the formal foundation of rule-based refinement and/or rule-based modification for the implementation of tool support. Due to the theory of Petri net transformations we have a precise description how rules and transformations work on Petri nets. Tool support is the main precondition for the practical use. The user should get tool support for defining and applying rules. The tool should assist the choice as well as the execution of rules and transformations.
- *Variations of the Development Process:* Another application area, where transformations are very useful, concerns variations in the development process. Often a development is not entirely unique, but variations of the same development process lead to variations in the desired models and resulting systems. These variations can be expressed by different rules yielding different transformations, that are used during the step-by-step development.

3. Emergency scenario case study

In this section we illustrate the main idea of net transformations by a case study of a pipeline emergency scenario where an unknown source of a natural gas leak is detected in a residential area1: A postal worker delivering mail in a residential street smells a strong odor of gas. She immediately notifies the fire department. A single engine company is dispatched by the fire department with four firefighters led by one company officer. At the scene, the postal worker meets the company officer and describes the problem. He calls the gas company and requests additional law enforcement officers to control traffic into the area. While three firefighters evacuate the homes in the immediate area and afterwards deny entry to this area, the forth one reads the gas indicator and detects that the gas is highest in front of a home located on 114 Maple Street. After electricity and gas lines are shut off to each home the fire department people stand by with fully charged hose lines and wait for the arrival of the gas company. The cooperative process enacted by the firefighter company is depicted as Petri net **PN1** in Fig. 1. This Petri net is decomposed into five parts corresponding to the team members described above, and in addition start as well as end activities. The union describes the gluing of the subnets along the interface given by the post domain places of transition *Start* (respectively pre domain places of transition *End).*

[1] www. pipelineemergencies.com

In this case the interface net consists of places only, so that the union corresponds to the usual place fusion of nets. But the general union construction allows having arbitrary subnets as interfaces.

In the following we show how Petri net transformations can be used in the case study before we present the basic concepts in Section 4. The three firefighters responsible for the evacuation process need more detailed information how to proceed. So the company officer gives the instruction that first of all the residents shall be notified of the evacuation. Afterwards the firefighters shall assist handicapped persons and guide all of them to the extent possible. To introduce the refinement of the *Evacuate homes*-transition into the Petri net **PN1** we provide the rule $r_{evacuate}$ depicted in the upper row of Fig. 2.

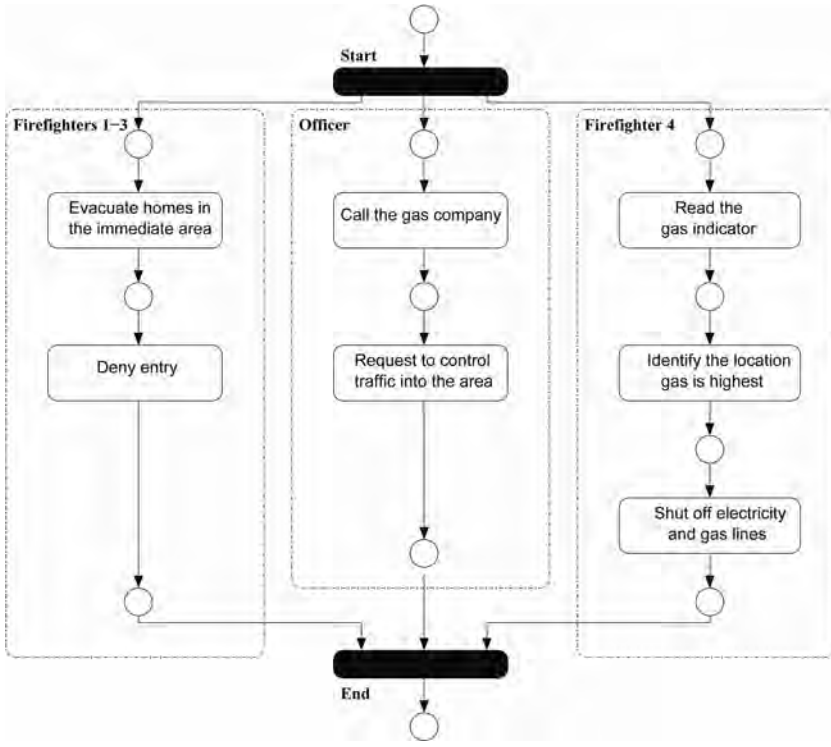


Fig. 1. Petri Net **PN1**

We show explicitly the direct transformation with rule *revacuate* from **Firefighters 1-3** (see Fig. 1) to **Firefighters 1-3'** in Fig. 2. The application of the rule is given as follows: the match morphism m is given by the obvious inclusion and identifies the relevant parts of the left hand side **L1** of rule $r_{evacuate}$ in **Firefighter 1-3.** In the first step we delete from **Firefighter 1-3** the *Evacuate homes*-transition and adjacent edges, but we preserve all places of **L1**, because they are also in **K1** and **R1,** leading to the context net **C** in Fig. 2. In the second step we glue together **C** and **R1** via **K1** by adding the transitions *Notify residents, Assist handicapped persons* and *Guide persons* together with their (new) environment to the context net **C** leading to **Firefighters 1-3'** in Fig. 2. Thus we obtain the direct transformation **Firefighters 1-3** $\overset{r_{evacuate}}{\Longrightarrow}$ **Firefighters 1-3'.**

Since the rule $r_{evacuate}$ and the direct transformation are preserving the interface of the corresponding union in Fig. 1, the interfaces are still available and can be used to construct a resulting net. The union theorem in Section 4 makes sure that this construction leads to the same result as if we would have applied the rule *revacuate* to the entire net **PN1** in Fig. 1. This is a typical example for compatibility of horizontal structuring (union) with vertical refinement (rule-based transformation).
After the problem identification the odor of gas grows stronger and the firefighter takes an additional reading of the gas indicator and informs the company officer about the result, so that the company officer is able to determine if the atmosphere in the area is safe, unsafe, or dangerous. To extend our process by these additional activities we use the rule $r_{analyse}$ in Fig. 3.

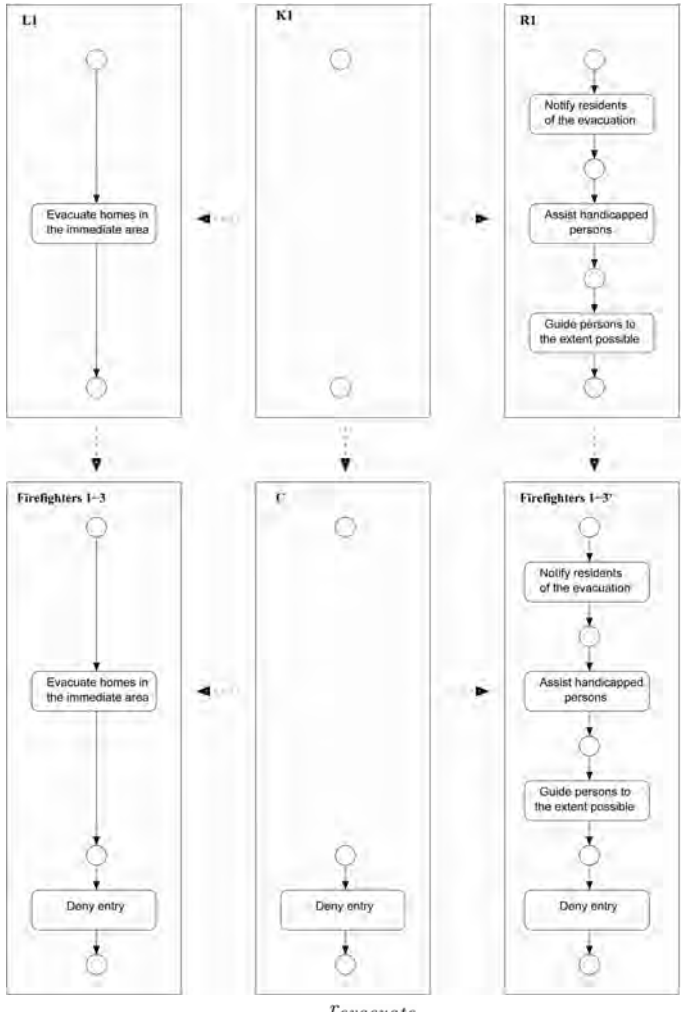


Fig. 2. Direct transformation **Firefighters 1-3** $\overset{r_{evacuate}}{\Longrightarrow}$ **Firefighters 1-3'**

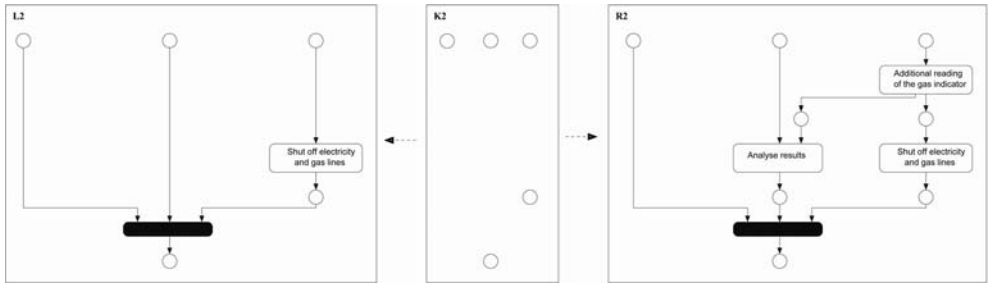


Fig. 3. Rule $r_{analyse}$

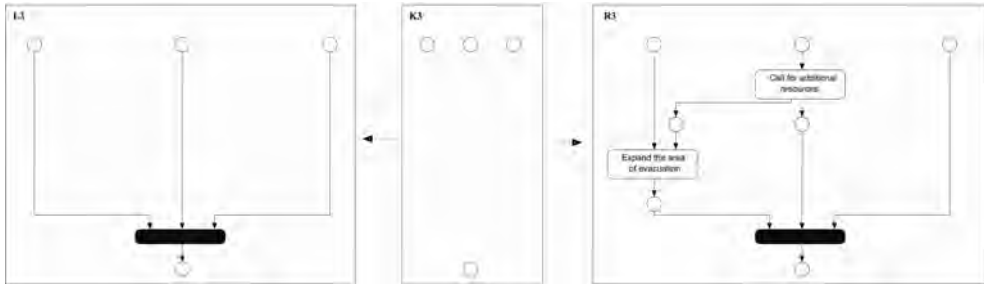


Fig. 4. Rule r_{expand}

Based on the additional results of the gas indicator the company officer analyses that the atmosphere in this area is over the lower explosive limit and thereby more dangerous than expected. He determines that the best course of action is to call for additional resources to maintain the isolation perimeter and expand the area of evacuation as a precaution. Here, we use rule r_{expand} depicted in Fig. 4 to extend the Petri net by the additional activities.
Summarizing, after the sequential application of the rules $r_{evacuate}$, $r_{analyse}$ and r_{expand} to the Petri net **PN1** in Fig. 1. we obtain the Petri net **PN4** in Fig. 5.

4. Concepts of Petri net transformations

Following up the informal overview in Section 2 we give in this section the precise definitions of the notions that we have already used in our case study. For notions and results beyond that we give a brief survey in Subsection 4.6 and refer to literature.
The concept of Petri net transformations [30, 8, 12, 7, 14] is a special case of high-level replacement systems. High-level replacement systems have been introduced in [9] as a categorical generalisation of the double-pushout approach to graph transformation, short DPO-approach. The theory of high-level replacement systems can be successfully employed not only to graph transformation, but also to other areas as Petri nets (see [9]). This leads to the concept of Petri net transformations as an instantiation of high-level replacements systems. In the following we explicitly present the resulting concepts of Petri net transformations for the case of place/transition nets.

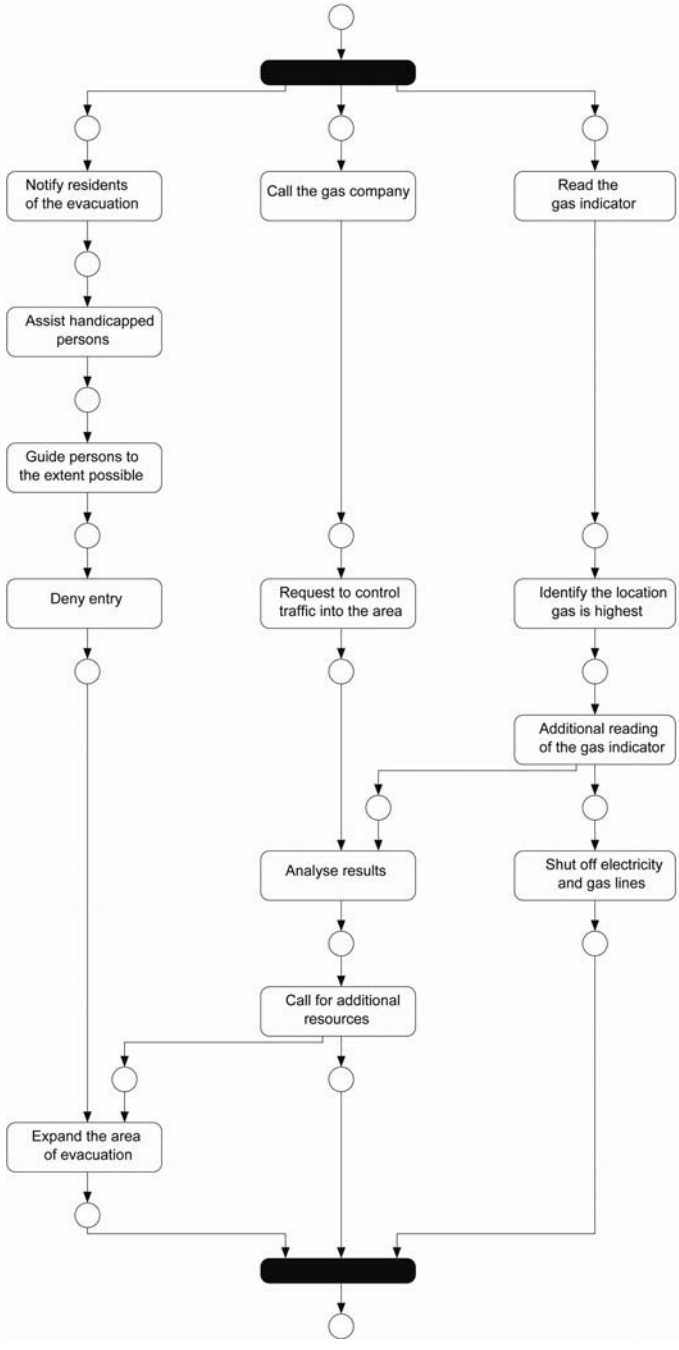


Fig. 5. Petri net **PN4**

4.1 Place/transition nets and net morphisms

Let us first present a notation of place/transition net that is suitable for our transformation approach. We assume that the nets are given in the algebraic style as introduced in [21]. A place/transition net $N = (P, T, pre, post)$ is given by the set of places P, the set of transitions T, and two mappings $pre, post : T \to P^{\oplus}$, the pre-domain and the post-domain,

$$T \underset{post}{\overset{pre}{\rightrightarrows}} P^{\oplus},$$

where $P^{\oplus}$ is the free commutative monoid over P that can also be considered as the set of finite multisets over P. The pre- (and post-) domain function maps each transition into the free commutative monoid over the set of places, representing the places and the arc weight of the arcs in the pre-domain (respectively in the post-domain). For finite P, an element $w \in P^{\oplus}$ can be presented as a linear sum $w = \sum_{p \in P} \lambda_p \cdot p$ with $\lambda_p \in$ N or as a function $w : P \to$ N. In the infinite case we have to require that $\lambda_p \neq 0$ only for finitely many $p \in P$ that means the corresponding $w : P \to$ N has finite support.

In the net **L3** in Fig. 4, T consists of one transition t and P of four places, where p_1, p_2, p_3 are shown above and p_4 below of t. The function $pre : T \to P^{\oplus}$ and $post : T \to P^{\oplus}$ are defined by $pre(t) = p_1 \oplus p_2 \oplus p_3$ and $post(t) = p_4$, respectively.

Based on the algebraic notion of Petri nets we use simple homomorphisms that are generated over the set of places. These morphisms map places to places and transitions to transitions. A morphism $f : N_1 \to N_2$ between two place/transition nets $N_1 = (P1, T1, pre1, post1)$ and $N_2 = (P_2, T_2, pre_2, post_2)$ is given by $f = (f_P, f_T)$ with mappings $f_P : P1 \to P_2$ and $f_T : T1 \to T_2$ that $pre_2 \circ f_T = f_P^{\oplus} \circ pre1$ and $post_2 \circ f_T = f_P^{\oplus} \circ post_1$. These conditions ensure that the pre-domain as well as the post-domain of a transition are preserved, so that, even if places may be identified, the number of tokens that are taken remains the same. Note that the extension $f_P^{\oplus} : P_1^{\oplus} \to P_2^{\oplus}$ of $f_P : P_1 \to P2$ is defined by $f_P^{\oplus}(\sum_{p \in P_1} \lambda_p \cdot p) = \sum_{p \in P_1} \lambda_p \cdot f_P(p)$. The morphism $f = (f_P, f_T)$ is called injective, if f_P and f_T are injective. The diagram schema for net morphisms is given in the following diagram.

$$\begin{array}{ccc} T_1 & \underset{post_1}{\overset{pre_1}{\rightrightarrows}} & P_1^{\oplus} \\ {\scriptstyle f_T}\downarrow & & \downarrow{\scriptstyle f_P^{\oplus}} \\ T_2 & \underset{post_2}{\overset{pre_2}{\rightrightarrows}} & P_2^{\oplus} \end{array}$$

Several examples of net morphisms can be found in Fig. 2 where the horizontal and vertical arrows denote injective net morphisms.

4.2 Rules and transformations

The formal definition of rules and transformations is based on concepts of the following category **PT.** The category **PT** consists of place/transition nets as objects and place/transition net morphisms as morphisms. In order to formalise rules and transformations for nets we first state the construction of pushouts in the category **PT** of place/transition nets. For any span of morphisms $N_1 \leftarrow N_0 \to N_2$ the pushout can be

constructed and means intuitively the gluing of nets N_1 and N_2 along N_0. The construction is based on the pushouts for the sets of transitions and places in the category **Set**. In the category **Set** of sets and functions the pushout object D is given by the quotient set $D = B + C/\equiv$, short $D = B +_A C$, where $B + C$ is the disjoint union of B and C and $\equiv$ is the equivalence relation generated by $f(a) \equiv g(a)$ for all $a \in A$. In fact, D can be interpreted as the gluing of B and C along A: Starting with the disjoint union $B + C$ we glue together the elements $f(a) \in B$ and $g(a) \in C$ for each $a \in A$. Given the morphisms $f : N_0 \to N_1$ and $g : N_0 \to N_2$ then the pushout N_3 in the category **PT** with the morphisms $f' : N_2 \to N_3$ and $g' : N_1 \to N_3$ is constructed (see diagram below) as follows:

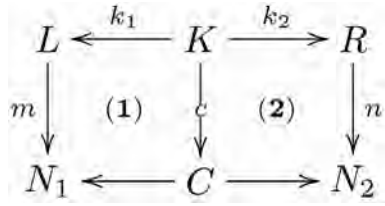
- $T_3 = T_1 +_{T_0} T_2$ with f'_T and g'_T as pushout of f_T and g_T in **Set**.
- $P_3 = P_1 +_{P_0} P_2$ with f'_P and g'_P as pushout of f_P and g_P in **Set** as well.
- $pre_3(t) = \begin{cases} [pre_1(t_1)] & ; \text{ if } g'_T(t_1) = t \\ [pre_2(t_2)] & ; \text{ if } f'_T(t_2) = t \end{cases}$
- $post_3(t) = \begin{cases} [post_1(t_1)] & ; \text{ if } g'_T(t_1) = t \\ [post_2(t_2)] & ; \text{ if } f'_T(t_2) = t \end{cases}$

$$\begin{array}{ccc} N_0 & \xrightarrow{f} & N_1 \\ {\scriptstyle g}\downarrow & (=) & \downarrow{\scriptstyle g'} \\ N_2 & \xrightarrow[f']{} & N_3 \end{array}$$

Two examples of the pushout construction of nets are depicted in Fig. 2. We have the embedding of **K1** into **L1** and **C**. The pushout describes the gluing of the nets **L1** and **C** along the two places of the interface **K1**. Hence we have the pushout **L1** $+_{K1}$ **C** **=Firefighters 1-3** on the left hand side of Fig. 2. Similarly, we have the pushout **R1** $+_{K1}$ **C** **=Firefighters 1-3'** on the right hand side of Fig. 2.

Since rule application always involves the construction of two pushouts, we speak of the double-pushout (DPO) approach to graph and net transformation, where transformation rules describe the replacement of the left-hand side net by the right-hand side net in the presence of an interface net.

- A rule $r = (L \xleftarrow{k_1} K \xrightarrow{k_2} R)$ consists of place/transition nets L, K and R, called left-hand side, interface and right-hand side net respectively, and two injective net morphisms $K \xrightarrow{k_1} L$ and $K \xrightarrow{k_2} R$.
- Given a rule $r = (L \xleftarrow{k_1} K \xrightarrow{k_2} R)$, a direct transformation $N_1 \overset{r}{\Longrightarrow} N_2$ from N_1 to N_2 is given by two pushout diagrams (1) and (2) in the following diagram. The morphisms $m : L \to N_1$ and $n : R \to N_2$ are called match and comatch, respectively. The net C is called pushout complement or the context net.



The illustration of a transformation can be found for our case study in Fig. 2, where the rule $r_{evacuate}$ is applied to the net **Firefighters 1-3** with match *m*. As explained above, the first pushout denotes the gluing of the nets **L1** and **C** along the net **Kl** resulting in the net **Firefighters 1-3.** The second pushout denotes the gluing of the nets **R1** and **C** along the net **Kl** resulting in the net **Firefighters 1-3'.**

4.3 Gluing condition and context nets

Given a rule *r* and a match *m* as depicted in the diagram above, then we construct in the first step the pushout complement *C* provided that a suitable gluing condition holds. This leads to the pushout (1) in the diagram above. In the second step we construct the pushout of *c* and *k2* leading to *N2* and the pushout (2) in the diagram above.

Intuitively the gluing condition makes sure that we can construct a context net *C,* also called pushout complement, from rule *r* and match *m* such that the gluing $C +_K L$ of *C* and *L* along *K* is equal to the net N*1*. Formally we have to require that dangling points and identification points are gluing points in the following sense:

Gluing Condition for Nets: $DP \cup IP \subseteq GP$, where the gluing points *GP,* dangling points *DP* and the identification points *IP* of *L* are defined by

- $GP = k_1(P_K \cup T_K)$,
- $DP = \{p \in P_L | \exists t \in (T_1 \setminus m_T(T_L)) : m_P(p) \in pre_1(t) \oplus post_1(t)\}$, and
- $IP = \{p \in P_L | \exists p' \in P_L : p \neq p' \wedge m_P(p) = m_P(p')\}$
  $\cup \{t \in T_L | \exists t' \in T_L : t \neq t' \wedge m_T(t) = m_T(t')\}$.

Now the pushout complement *C* is constructed by:

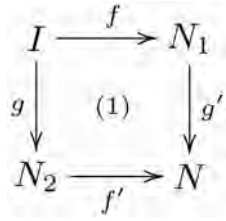
- $P_C = (P_1 \setminus m_P(P_L)) \cup m_P(k_{1P}(P_K))$
- $T_C = (T_1 \setminus m_T(T_L)) \cup m_T(k_{1T}(T_K))$
- $pre_C = pre_{1|T_C}$ and $post_C = post_{1|T_C}$

Note that the pushout complement C leads to the pushout (1) in the diagram above and that it is unique up to isomorphism.

In our case study in Section 3, the gluing condition is satisfied in the direct transformation in Fig. 2 since the match is injective and places are not deleted by the rule $r_{evacuate}$. In fact, the dangling points *DP* of the match in Fig. 2 are given by one place of **L1**, while the gluing points *GP* consists of all places in **L1.** The set of identification points *IP* is empty, because the match is injective, hence we have $DP \cup IP \subseteq GP$.

4.4 Union construction

The union of two Petri nets sharing a common subnet, that may be empty, is defined by the pushout construction for nets. The union of place/transition nets N1, N2 sharing an interface net I with the net morphisms $f : I \to$ N1 and $g : I \to$ N2 is given by the pushout diagram (1) below. Subsequently we use the short notation $N = N1 + I\ N2$ or $N1; N2 > \overset{I}{\Longrightarrow} N$.

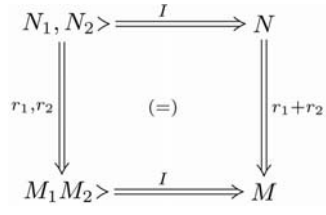


In our example in Fig. 1 we can use the union construction several times to describe the net **PN1** as the composition of five different subnets given by **Firefighters 1-3, Officer, Firefighter 4, Start** and **End**. The interface nets I are given by the intersection of the corresponding nets.

4.5 Union theorem

The Union Theorem states the compatibility of union and net transformations in the following sense: A union of two nets followed of a parallel transformation of the united nets yields the same result as two transformations of the original two nets followed by a union of the two transformed nets.

Given a union $N1 + I\ N2 = N$ and net transformations $N1 \overset{r_1}{\Longrightarrow} M_1$ and $N2 \overset{r_2}{\Longrightarrow} M_2$ then we have a parallel rule $r_1+r_2 = (L_1+L_2 \leftarrow K_1+K_2 \rightarrow R_1+R_2)$, where $L_1 + L_2$, $K_1 + K_2$ and $R_1 + R_2$ are disjoint unions of the respective nets of rules r_1 and r_2, and a parallel net transformation $N \overset{r_1+r_2}{\Longrightarrow} M$. Then $M = M_1 + I\ M_2$ is the union of M_1 and M_2 with the shared interface I, provided that the given net transformations preserve the interface I. The Union Theorem is illustrated in the following diagram and especially stated and proven in [22]:



Note that the compatibility requires an independence condition stating that nothing from the interface net I may be deleted by one of the transformations of the subnets.

This allows in Section 3 to apply either the rules $r1 = r_{evacuate}$ and $r2 = r_{analyse}$, respectively, to $N1$ =**Firefighters 1-3** in Fig. 1 and $N2$ constructed as union in four steps of the nets **Officer, Firefighter 4, Start** and **End**, or in parallel to the union $N = N1 +_I N2$, where I consists of two places which are preserved by both transformations $N1 \overset{r_1}{\Longrightarrow} M1$ and $N2 \overset{r_2}{\Longrightarrow} M_2$. This allows

to obtain the same net M by union $M = M_1 +_I M_2$ and by transformation $N \overset{r_1+r_2}{\Longrightarrow} M$. Finally, applying rule $r_3 = r_{expand}$ to M leads to the net PN4 in Fig. 5.

4.6 Further results

We briefly introduce the main net classes which have been studied up to now and subsequently present some main results.

- Place/transition nets in the algebraic style have already been introduced in Subsection 4.1. In [11, 17, 10] we have transferred these results to place/transition systems, where a place/transition system is a place/transition net with an initial marking.
- Coloured Petri nets [18, 19, 20] are high-level nets combining P/T nets and ML expressions for data type definitions. They are very popular due to the tool CPN-tools [5].
- Algebraic high-level nets are available in quite a few different notions e.g. [28, 25]. We use a notion that reflects the paradigm of abstract data types into signature and algebra. An algebraic high-level net (as in [25]) is given by $N = (SPEC,P,T,pre,post,cond,A)$, where $SPEC = (S,OP,E;X)$ is an algebraic specification in the sense of [13] with additional variables X not occurring in E, P is the set of places, T is the set of transitions, pre,post : $T \to (T_{OP}(X) \times P)^{\oplus}$ are the pre- and post-domain mappings, cond : $T \to P_{fin}(EQNS(SIG, X))$ are the transition guards, and A is a SPEC algebra.

Horizontal Structuring Union and fusion are two categorical structuring constructions for place/transition nets that merge two subnets (fusion) or two different nets (union) into one.
The union has been introduced in the previous subsection. Now let us consider the fusion: Given a net F that occurs in two copies in the net $N1$, represented by two morphisms $F \underset{f'}{\overset{f}{\rightrightarrows}} N_1$, the fusion construction leads to a net where both occurrences of F in N_1 are merged. If F consists of places p1,...,pn then each of the places occurs twice in net $N1$, namely as $f(p1)$, ..., $f(p_n)$, and $f'(p1)$,..., $f'(p_n)$. N_2 is obtained from the net $N1$ by fusing both occurrences $f(pi)$ and $f'(pi)$ of each place pi for $1 \leq i \leq n$.
The Union Theorem has been presented in the previous subsection. The Fusion Theorem [23] is expressed similarly: Given a rule r and a fusion $F \rightrightarrows N_1$ then we obtain the same result whether we derive first $N_1 \overset{r}{\Longrightarrow} N_1'$ and then construct the fusion $F \rightrightarrows N_1'$ resulting in N_2' or whether we construct the fusion $F \rightrightarrows N_1$ first, resulting in N_2, and then perform the transformation step $N_2 \overset{r}{\Longrightarrow} N_2'$. Similar to the Union Theorem, a certain independence condition is required. Both theorems state that Petri net transformations are compatible with the corresponding structuring technique under suitable independence conditions. In short these conditions guarantee that the interface net I and respectively the fusion net F are preserved by all net transformations.
Interleaving and Parallelism We are able to realize model interleaving and parallelism of net transformations. The Local Church- Rosser Theorem states a local confluence in the sense of formal languages corresponding to interleaving. The required condition of parallel independence means that the matches of both rules overlap only in parts that are not deleted. Sequential independence means that those parts created or used by the first transformation step are not used or deleted in the second step, respectively. The Parallelism Theorem states that sequential or parallel independent transformations can be carried out either in arbitrary sequential order or in parallel. In the context of step-by-step development these theorems are important as they provide conditions for the independent

development of different parts or views of the system. More details on horizontal structuring or parallelism are given in [25] and [23].

Refinement Rule-based refinement comprises the transformation of Petri nets using rules while preserving certain net properties. For Petri nets the desired properties of the net model can be expressed e.g in terms of Petri nets (as liveness, boundedness etc.), in terms of logic (e.g. temporal logic, logic of actions etc.), in terms of relation to other models (e.g. bisimulation, correctness etc.), and so on.

For place/transition nets, algebraic high-level nets and Coloured Petri nets the most important results for rule-based refinement are presented in Table 1. For more details see [27].

Notion/Results	PT-nets	AHL-nets	CPNs
Rules, Transformations	√	√	√
Safety property preserving transformations with			
transition-gluing morphisms	√	√	√
place-preserving morphisms	√	√	√
Safety property introducing transformations	√	√	√
Liveness preserving transformations	√	?	?
Liveness introducing transformations	√	?	?
Local Church Rosser I + II Theorem	√	√	√
Parallelism Theorem	√	√	√
Union	√	√	√
Fusion	√	√	√
Union Theorem	√	√	√
Fusion Theorem	√	√	√

Table 1. Achieved results

5. Conclusion

The main idea of Petri net transformations is to extend the classical theory of Petri nets by a rule-based technique that allows to model the changes of the Petri net structure.

There have been already a few approaches to describe transformations of Petri nets formally (e.g. in [2, 3, 31, 6, 32]). The intention has been mainly on reduction of nets to support verification, and not on the software development process as in our case. This use of transformations has been one of the main focus areas of the DFG-Research group *Petri Net Technology*. There are some large studies in various application areas as medical information

systems [15], train control systems [26], or as sketched in this paper in emergency scenarios. These case studies clearly show the advantages using net transformation in system development and the practical use of the results stated in Table 1. Although the area of Petri net transformations is already well-established, there are many promising directions for further research to follow, for example:

- Transfer to other net classes
  There is a large variety of Petri net classes, and in principle the idea of Petri net transformation is applicable to all of them. The concept of transformation we have employed is an algebraic one, so the use of algebraic approaches to Petri nets is more suggesting. Algebraic higher-order nets [16] have been recently developed and are one of the promising targets to transfer the idea of transformations to. These nets extend algebraic high-level nets as they are equipped with a higher-order signature and algebra. This allows most interesting applications and supports structure flexibility and system adaptability in an extensive way.
- Reconfigurable place/transitions systems
  In [17], the concept of reconfigurable place/transition (P/T) systems has been introduced that is most important to model changes of the net structure while the system is kept running. In detail, a reconfigurable P/T-system consists of a P/T-system and a set of rules, so that not only the follower marking can be computed but also the structure can be changed by rule application to obtain a new P/T-system that is more appropriate with respect to some requirements of the environment. Moreover these activities can be interleaved. In [11] we have continued our work by transferring the results of local Church-Rosser which are well known for term rewriting and graph and net transformations (see [30, 7, 10]) to the consecutive evolution of a P/T-system by token firing and rule applications. In more detail, we assume that a given P/T-system represents a certain system state. The next evolution step can be obtained not only by token firing, but also by the application of one of the rules available. Hence, we have presented conditions for (co-)parallel and sequential independence, such that each of these evolution steps can be postponed after the realization of the other, yielding the same result and, analogously, they can be performed in a different order without changing the result.
- Component technology
  Components present an advanced paradigm for the structuring of complex systems and have been advocated in the recent years most strongly. Components that use Petri nets for the specification of the interfaces and the component body have been defined in [24]. There are three nets that represent the import, the export and the body of the component. The export is an abstraction of the body and the import is embedded into the body. There are two operations: the hierarchical composition and the union of components. Unfortunately, up to now there is no transformation concept in the sense of graph and net transformation. Based on net transformations the transformation of the import, the export and the body can be defined straightforward.
- Tool support
  The practical use of graph transformation is supported by several tools. The algebraic approach to graph transformation is especially supported by the graph transformation environment AGG (see [1]). A tool for net transformations using the graph

transformation engine AGG has been developed recently [29] as an Eclipse plug-in to support a special class of reconfigurable P/T-systems.

6. References

[1] AGG Homepage, http://tfs.cs.tu-berlin.de/agg.

[2] G. Berthelot. Checking Properties of Nets using Transformations. In *Advances in Petri Nets,* volume 222 of *LNCS,* pages 19-40. Springer, 1986.

[3] G. Berthelot. Transformations and Decompositions of Nets. In *Advances in Petri Nets,* volume 254 of *LNCS,* pages 359-576. Springer, 1987.

[4] P. Bottoni, F. De Rosa, K. Hoffmann, and M. Mecella. Applying Algebraic Approaches for Modeling Workflows and their Transformations in Mobile Networks. *Mobile Information Systems,* 2(1):51—76, 2006.

[5] CPN Tools Homepage. http://wiki.daimi.au.dk/cpntools/_home.wiki.

[6] R. David and H. Alia, editors. *Petri Nets and Grafcet.* Prentice Hall (UK), 1992.

[7] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[8] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools.* World Scientific, 1999.

[9] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct, in Comp. Science,* 1:361-404, 1991.

[10] H. Ehrig, K. Hoffmann, U. Prange, and J. Padberg. Formal Foundation for the Reconfiguration of Nets. Technical Report Technical Report 2007-02, Technical University Berlin, Fak. IV, 2007.

[11] H. Ehrig, J. Padberg K. Hoffmann, U. Prange, and C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In *Proc. Application and Theory of Petri Nets (ATPN),* volume 4546 of *LNCS,* pages 104-123, 2007.

[12] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol 3: Concurrency, Parallelism and Distribution.* World Scientific, 1999.

[13] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[14] H. Ehrig and J. Padberg. Graph Grammars and Petri Net Transformations. In *Lectures on Concurrency and Petri Nets, Special Issue Advanced Course PNT,* volume 3098 of *LNCS,* pages 496-536. Springer, 2004.

[15] C. Ermel, J. Padberg, and H. Ehrig. Requirements Engineering of a Medical Information System Using Rule-Based Refinement of Petri Nets. In *Proc. Integrated Design and Process Technology (IDPT),* volume 1, pages 186— 193. Society for Design and Process Science, 1996.

[16] K. Hoffmann. *Formal Approach and Applications of Algebraic Higher Order Nets.* PhD thesis, Technical University Berlin, 2005.

[17] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In *Proc. Application and Theory of Petri Nets (ATPN),* volume 3536 of *LNCS,* pages 268-288. Springer, 2005.

[18] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use,* volume 1: Basic Concepts, of *EATCS Monographs in Theoretical Computer Science.* Springer, 1992.

[19] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use,* volume 2: Analysis Methods of *EATCS Monographs in Theoretical Computer Science.* Springer, 1995.

[20] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use,* volume 3: Practical Use of *EATCS Monographs in Theoretical Computer Science.* Springer, 1997.

[21] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation,* 88(2):105-155, 1990.

[22] J. Padberg. *Abstract Petri Nets: A Uniform Approach and Rule-Based Refinement.* PhD thesis, Technical University Berlin, 1996. Shaker Verlag.

[23] J. Padberg. Categorical Approach to Horizontal Structuring and Refinement of High-Level Replacement Systems. *Applied Categorical Structures,* 7(4):371-403, 1999.

[24] J. Padberg. Basic Ideas for Transformations of Specification Architectures. In *Proc. Workshop on Software Evolution through Transformations (SET 02),* volume 74 of *ENTCS,* 2002.

[25] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science,* 5(2):217-256, 1995.

[26] J. Padberg, P. Schiller, and H. Ehrig. New Concepts for High-Level Petri Nets in the Application Domain of Train Control. In *Proc. Symposium on Transportation Systems,* pages 153-160, 2000.

[27] J. Padberg and M. Urbasek. Rule-Based Refinement of Petri Nets: A Survey. In *Proc. Petri Net Technology for Communication-Based Systems,* volume 2472 of *LNCS,* pages 161-196. Springer, 2003.

[28] W. Reisig. Petri Nets and Algebraic Specifications. *Theoretical Computer Science,* 80:1-34, 1991.

[29] RON Editor Homepage, http://tfs.cs.tu-berlin.de/roneditor/.

[30] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[31] Vanio M. Savi and Xiaolan Xie. Liveness and Boundedness Analysis for Petri Nets with Event Graph Modules. In *Proc. Application and Theory of Petn Nets (ATPN),* volume 254 of *LNCS,* pages 328-347. Springer, 1992.

[32] W.M.P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets,* volume 1248 of *LNCS,* pages 407-426. Springer, 1997.

2

Modelling and Analysis of Real-Time Systems with RTCP-Nets

Marcin Szpyrka
AGH University of Science and Technology, Krakow
Poland

1. Introduction

RTCP-nets (Real-Time Coloured Petri nets, (Szpyrka 2006a), (Szpyrka & Szmuc 2006c)) are a subclass of timed coloured Petri nets (CP-nets, (Jensen 1992-1997)) defined for modelling and analysis of real-time systems. In comparison to timed CP-nets, RTCP-nets use a different time model, transitions' priorities and they are forced to fulfil some structural restrictions. These characteristics of RTCP-nets enable designers direct modelling of elements typical for concurrent programming (e.g. in Ada programming language, (Barnes 2006)), such as task priorities, timeouts, etc.

Formal methods (Cheng 2002) are used in the development of embedded systems for design, specification, validation, and verification of such systems. The use of formal methods can reduce the amount of testing and ensure more dependable products (Sommerville 2004). Especially, this is very important for safety-critical systems that may result in injury, loss of life or serious environmental damage upon their failure. A wide class of real time systems perform on the basis of a set of rules, which are used to compute outputs in response to current state of inputs that are monitored in such system environment. This set of rules specified in the analysis phase as functional requirements may be formally described, and then incorporated into the system model.

The presented approach uses RTCP-nets as modelling language for safety-critical systems. The modifications defining this subclass were introduced in order to improve modelling and verification means in the context of analysis and design of embedded systems. Especially, this technique has mostly been concerned with relatively small, critical kernel systems. RTCP-nets have been also prepared for modelling of embedded systems incorporating rule-based systems. A rule-based system in decision table form can be simply included into a model.

Another advantage of RTCP-nets is relatively simple transformation from a formal model into an implementation in Ada 2005 programming language. Such an implementation is done with the use of so-called Ravenscar profile (Burns et al. 2003). The profile is a subset of Ada language. It has been defined to allow implementation of safety-critical systems in Ada.

The goal of the chapter is to present the most important parts of the RTCP-nets theory and to describe the possibilities of practical applications of the nets. The chapter is organized as follows. The first section deals with a formal definition of RTCP-nets. The behaviour of the nets is presented in details so as to emphasize the differences between RTCP-nets and CP-nets. This part of the chapter is illustrated with an example of a non-hierarchical RTCP-net

(an example of a simple train protection systems).
The second section describes the analysis methods. It focuses on coverability graphs that are typical for RTCP-nets. If a net is strongly bounded, it is possible to construct a finite coverability graph that represents the set of all reachable states regardless of the fact the set is finite or infinite. Such a graph contains only one node for each equivalence class of the coverability relation. Not only can one use such a graph for the analysis of typical Petri nets' properties such as boundedness, liveness or fairness, but it also may be used for verification of timing properties, which are very important for most real-time embedded systems.
The last section deals with practical aspects of modelling with RTCP-nets. To speed up and facilitate drawing of more complex models the so-called *canonical form* of hierarchical RTCP-nets has been defined. The canonical form is shortly described in this section and an RTCP-net model of a real size railway traffic management system for a train station is presented to illustrate the possibilities of modelling with the nets.
The chapter is concluded with a short summary that describes possibilities of semiautomatic generation of an Ada 2005 source code from RTCP-nets models in canonical form.

2. RTCP- nets - basic notions

The definition of RTCP-nets is based on the definition of non-hierarchical timed CP-nets presented in (Jensen 1992-1997), but a few differences between timed CP-nets and RTCP-nets can be pointed out:

- Each transition has a priority value attached. The use of priorities allows direct modelling of deterministic choice.
- The set of arcs is defined as a relation due to the fact that multiple arcs are not allowed. Each arc has two expressions attached: a weight expression and a time expression. For any arc, each evaluation of the arc weight expression must yield a single token belonging to the type (colour) that is attached to the corresponding place; and each evaluation of the arc time expression must yield a non-negative rational value.
- The time model used by RTCP-nets differs from the one used by timed CP-nets. Time stamps are attached to places instead of tokens. Any positive value of a time stamp describes how long a token in the corresponding place will be inaccessible for any transition. A token is accessible for a transition, if the corresponding time stamp is equal to or less than zero. For example, if the stamp is equal to -3, it means the token is 3 time-units old. It is possible to specify how old a token should be so that a transition may consume it.

For any variable v, $\mathcal{T}(v)$ will be used to denote the *type* of the variable i.e. the set of all admissible values, the variable can be associated with. Let x be an expression. $\mathcal{V}(x)$ will denote the set of all variables in the expression x, and $\mathcal{T}(x)$ will denote the *type* of the expression, i.e. the set of all possible values that can be obtained by evaluating of the expression. For any given set of variables V, the type of the set of variables is defined as follows: $\mathcal{T}(V) = \{\mathcal{T}(v): v \in V\}$.
Let *Bool* denote the boolean type (containing the elements *{false,true}*, and having the standard operations from propositional logic). Let $\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{Q}$, $\mathbb{Q}^+$ denote the set of natural, rational and non-negative rational numbers respectively. For an arc *a*, *P(a)* and *T(a)* will be used to denote the *place node* and the *transition node* of the arc, respectively.
Definition 1. An *RTCP-net* is a tuple $\mathcal{N} = (\Sigma, P, T, A, C, G, I, E_M, E_S, M_0, S_0)$ satisfying the following requirements.

1. Σ is a non-empty finite set of non-empty *types (colour sets).*
2. P is a non-empty finite set of *places.*
3. T is a non-empty finite set of *transitions* such that $P \cap T = \emptyset$
4. $A \subseteq (P \times T) \cup (T \times P)$ is a *set of arcs.*
5. $C: P \to \Sigma$ is a fype *function,* which maps each place to its type.
6. G is a guard *function,* which maps each transition to an expression such that: $\forall t \in T: \mathcal{T}(G(t)) \subseteq Bool \wedge \mathcal{T}(\mathcal{V}(G(t))) \subseteq \Sigma$.
7. $I: T \to \mathbb{N} \cup \{0\}$ is a *priority function,* which maps each transition to a non-negative integer called *transition priority.*
8. E_M is an *arc expression function,* which maps each arc to a weight expression such that: $\forall a \in A: \mathcal{T}(E_M(a)) \subseteq C(P(a)) \wedge \mathcal{T}(\mathcal{V}(E_M(a))) \subseteq \Sigma$.
9. E_S is an arc time *expression function,* which maps each arc to a time expression such that: $\forall a \in A: \mathcal{T}(E_S(a)) \subseteq \mathbb{Q}^+ \cup \{0\} \wedge \mathcal{T}(\mathcal{V}(E_S(a))) \subseteq \Sigma$,
10. M_0 is an *initial marking,* which maps each place to a multiset $M_0(p) \in 2^{C(p)^*}$, where $2^{C(p)^*}$ denotes the set of all multisets over the set *C(p).*
11. $S_0: P \to \mathbb{Q}$ is an *initial time stamp function,* which maps each place to a rational value called *initial time stamp.*

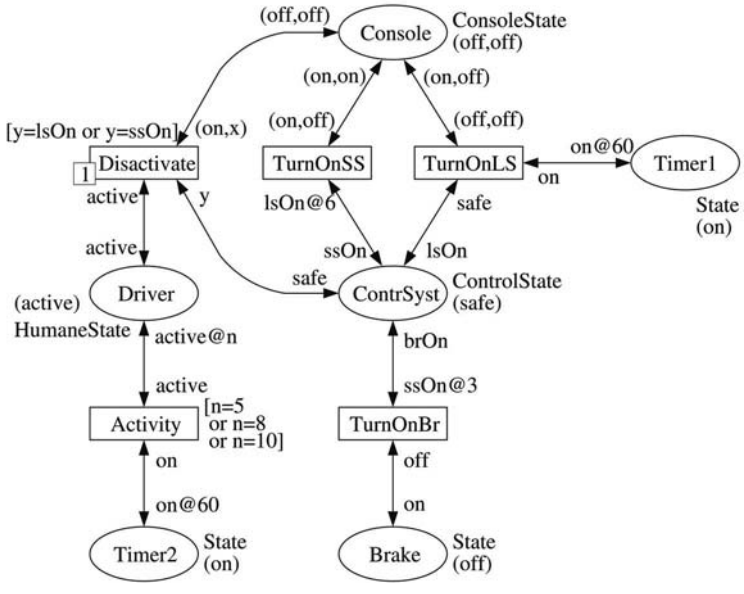


Fig. 1. Model of a simple ATS system

A model of a simple Automatic Train Stop (ATS) system is used to introduce main features of RTCP-nets. In the ATS system, a light signal is turned on every 60 seconds to check whether the driver controls the train. If the driver fails to acknowledge the signal within 6 seconds, a sound signal is turned on. Then, if the driver does not disactivate the signals within 3 seconds, using the acknowledge button, the emergency brakes are applied automatically to stop the train. A model of such a system is shown in Fig. 1. More information on using RTCP-nets for modelling train protection systems can be found in (Szpyrka & Szmuc 2006b).

The RTCP-net presented in Fig. 1 contains six places: *ContrSyst* (the control element of the ATS system), *Console* (to display warning signals), *Brake, Driver, Timerl* and *Timer2;* and five transitions: *TurnOnLS* (turn on light signal), *TurnOnSS* (turn on sound signal), *TurnOnBr* (turn on brake), *Disactivate* (driver disactivates warning signals) and *Activity* (to introduce into model some delays of the driver response). Initial markings are placed into parenthesis and initial time stamps equal to 0 are omitted. The transition's *Disactivate* priority is equal to 1, while other transition's priorities are equal to 0. The weight and time expressions are separated by the @ sign. If a time expression is equal to 0 it is omitted. Each arc with double arrows stands for a pair of arcs.

Definition 2. A *marking* of an RTCP-net $\mathcal{N}$ is a function M defined on the set of places P, such that: $\forall p \in P: M(p) \in 2^{C(p)^*}$. A *time stamp function* is a function S defined on the set of places P, such that: $\forall p \in P: S(p) \in \mathbb{Q}$.

If we assume that P is ordered set, both a marking M and a time stamp function S can be represented by vectors with $|P|$ entries. Therefore, the term a *time stamp vector* (or a *time vector)* will be used instead of a time stamp function.

Definition 3. A state of an RTCP-net is a pair (M, S), where M is a marking and S is a time stamp vector. The *initial state* is the pair (M_0, S_0).

Let's consider the net presented in Fig. 1 and let the set of places be ordered as follows P = {*ContrSyst, Timer1, Console, Brake, Driver, Timer2*}. The initial state of the considered net is as follows:

$$M_0 = \text{(safe, on, (off, off), off, active, on)},$$
$$S_0 = (0,0,0,0,0,0). \tag{1}$$

Let $X = P \cup T$ denote the set of all nodes of an RTCP-net and $x \in X$. *In(x)* and *Out(x)* denote the set of *input and output nodes* of the node *x,* i.e. $In(x) = \{y \in X: (y,x) \in A\}$ and $Out(x) = \{y \in X: (x,y) \in A\}$. Let $\mathcal{V}(t)$ be the set of variables that occur in the expressions of arcs surrounding the transition *t* and in the guard of the transition.

Definition 4. A *binding of* a transition $t \in T$ is a function b defined on $\mathcal{V}(t)$, such that: $\forall v \in \mathcal{V}(t): b(v) \in \mathcal{T}(v) \wedge G(t)_b = true$.

Intuitively, a binding of a transition t is a substitution that replaces each variable of $\mathcal{V}(t)$ with a value of the corresponding type, such that the guard evaluates to *true.* The set of all bindings of a transition t is denoted by $\mathcal{B}(t)$. $G(t)_b$ denotes the evaluation of the guard expression in the binding b. Similarly, $E_M(p,t)_b$ and $E_S(p,t)_b$ denote the evaluation of the weight and the time expression in the binding b, respectively.

Definition 5. A transition $t \in T$ is *enabled* in a state (M, S) in a binding b iff the following conditions hold:

$$\begin{aligned} &\forall p \in In(t): E_M(p,t)_b \in M(p) \wedge E_S(p,t)_b \leq -S(p), \\ &\forall p \in Out(t): S(p) \leq 0. \end{aligned} \tag{2}$$

and for any transition $t' \neq t$ that satisfies the above conditions in some binding $b' \in \mathcal{B}(t')$, $I(t') \leq I(t)$ or $In(t) \cap In(t') = Out(t) \cap Out(t') = \emptyset$.

It means that a transition is enabled if all input places contain suitable tokens and have suitable time stamps, all output places are accessible and no other transition with a higher priority strives for the same input or output places.

A transition $t \in T$ is enabled in a state *(M, S)* if it is enabled in the state *(M, S)* in one of its bindings. If a transition $t \in T$ is enabled in a state (M_1, S_1) in a binding b it may *fire*,

changing the state (M_1, S_1) to another state (M_2, S_2), such that $M_2(p) = M_1(p) - \{E_M(p,t)_b\} \cup \{E_M(t,p)_b\}$, ($\{E_M(x,y)_b\} = \emptyset$ if $(x,y) \notin A$), and

$$S_2(p) = \begin{cases} E_S(t,p)_b & \text{for } p \in Out(t), \\ 0 & \text{for } p \in In(t) - Out(t), \\ S_1(p) & \text{otherwise.} \end{cases} \tag{3}$$

In other words, if a transition fires, it removes one token from each input place, adds one token to each output place, sets time stamps of input places to 0 and sets time stamps of output places to values specified by time expressions of arcs leading from the transition to the places.

If a transition $t \in T$ is enabled in a state (M_1, S_1) in a binding b and a state (M_2, S_2) is derived from firing of the transition, then we write $(M_1, S_1) \xrightarrow{(t,b)} (M_2, S_2)$. The binding b will be omitted if it is obvious or redundant.

Two transitions *Activity* and *TurnOnLS* are enabled in the initial state. The first transition is enabled in three different bindings: $b_1 = (5/n)$ (the value of the variable *n* is equal to 5), $b_2 = (8/n)$ and $b_3 = (10/n)$, while the second one is enabled in the binding *b = ()* (a trivial binding). For example, the result of firing of the transition *TurnOnLS* in the initial state is the state (M_1, S_1), where:

$$\begin{aligned} M_1 &= (lsOn, on, (on, off), off, active, on), \\ S_1 &= (0, 60, 0, 0, 0, 0). \end{aligned} \tag{4}$$

A global clock is used to measure time. Every time the clock goes forward, all time stamps are decreased by the same value.

Definition 6. Let *(M, S)* be a state and $e = (1, 1, \ldots, 1)$ a vector with $|P|$ entries. The state (*M, S)* is changed into a state (*M', S')* by a passage *of time* $\tau \in \mathbb{Q}^+$, denoted by $(M,S) \xrightarrow{\tau} (M',S')$, iff $M = M'$ and the passage of time τ is possible, i.e., no transition is enabled in any state *(M, S")*, such that: $S'' = S - \tau' \cdot e$, for $0 \leq \tau' < \tau$ and $\tau' \in \mathbb{Q}^+$.

The result of firing of transitions *TurnOnLS* and *Activity* (in binding b_2) is the state *(M_2,S_2)*, where $M_2 = M_1$ and $S_2 = (0, 60, 0, 0, 8, 60)$. None transition is enabled in the state but it is possible a passage of time $\tau = 6$ that leads to the state (M_2, S_2'), where $S_2' = (-6, 54, -6, -6, 2, 54)$. A timeout occurs in this state. A token in the place *Console* is 6 seconds old (the driver did not response within 6 seconds), so the transition *TurnOnSS* will fire.

A *firing sequence* of an RTCP-net $\mathcal{N}$ is a sequence of pairs $\alpha = (t_1, b_1), (t_2, b_2), \ldots$, such that b_i is a binding of the transition t_i for $i = 1, 2, \ldots$ The firing sequence is *feasible* from a state (M_1, S_1) iff there exists a sequence of states such that:

$$(M_1, S_1) \xrightarrow{\tau_1} (M_1, S_1') \xrightarrow{(t_1,b_1)} (M_2, S_2) \xrightarrow{\tau_2} \ldots \xrightarrow{(t_n,b_n)} (M_{n+1}, S_{n+1}) \xrightarrow{\tau_{n+1}} \ldots \tag{5}$$

For the sake of simplicity, we will assume that there is at most one passage of time (sometimes equal to 0) between firings of two consecutive transitions. A firing sequence may be finite or infinite. The set of all firing sequences feasible from a state (*M, S)* is denoted by $\mathcal{L}(M, S)$.

A state (*M', S')* is *reachable* from a state (*M, S)* iff there exists a finite firing sequence α feasible from the state (*M, S)* and leading to the state (*M', S')*. In such case, we can also say that the marking *M'* is *reachable* from the marking *M*. The set of all states that are reachable

from *(M, S)* is denoted by $\mathcal{R}(M,S)$, while $\mathcal{R}(M)$ denotes the set of all markings reachable from the marking *M*.

3. Analysis of RTCP-nets

A major strength of Petri nets is their support for analysis of many properties and problems associated with concurrent systems. Three types of properties are distinguished for RTCP-nets: boundedness, liveness and timing ones.

Definition 7. Let an RTCP-net $\mathcal{N}$, a place $p \in P$, *a* multiset $X \in 2^{C(p)^*}$ and a non-negative integer *k* be given.

1. *k* is *upper integer bound* for p iff $\forall(M,S) \in \mathcal{R}(M_0,S_0)$: $|M(p)| \leq k$.
2. X is *upper multiset bound* for p iff: $\forall(M,S) \in \mathcal{R}(M_0,S_0)$: $M(p) \leq X$.

Lower bounds are defined analogously. A place p is said to be *bounded* if it has an upper integer bound. If the upper integer bound is equal to one, the place is said to be *safe*. A place p is said to be *strongly bounded* if it has a finite upper multiset bound. An RTCP-net $\mathcal{N}$ is said to be bounded if each place $p \in P$ has an upper integer bound. Safe and *strongly bounded* RTCP-nets are defined analogously.

An RTCP-net is *conservative* iff the number of tokens in the net remains constant. An RTCP-net is conservative with respect to a weighting vector $w = (w_1, w_2, \ldots, w_n)$, where $w_i > 0$, $i = 1,2,\ldots,n$, iff the weighted number of tokens remains constant, i.e. $\forall(M,S) \in \mathcal{R}(M_0,S_0)$: $\sum_{i=1}^{n} w_i|M(p_i)| = \sum_{i=1}^{n} w_i|M_0(p_i)|$.

The concept of liveness is closely related to the complete absence of deadlocks. Five different levels of liveness can be defined for Petri nets (see (Murata 1989)).

Definition 8. Let an RTCP-net $\mathcal{N}$ be given. A transition $t \in T$ is said to be:

(0) *dead* ($\mathcal{L}0$-*live*) if t does not appear in any firing sequence in $\mathcal{L}(M_0,S_0)$.
(1) *potentially fireable* ($\mathcal{L}1$-*live*) if t appears at least once in some firing sequence in $\mathcal{L}(M_0,S_0)$.
(2) $\mathcal{L}2$-*live* if, given any positive integer k, t appears at least k times in some firing sequence in $\mathcal{L}(M_0,S_0)$.
(3) $\mathcal{L}3$-*live* if t appears infinitely often in some firing sequence in $\mathcal{L}(M_0,S_0)$.
(4) *live* ($\mathcal{L}4$-*live*) if t is $\mathcal{L}1$-live for each state $(M,S) \in \mathcal{R}(M_0,S_0)$.

An RTCP-net is said to be $\mathcal{L}k$-*live* if each transition of the net is $\mathcal{L}k$-*live*, $k = 0,\ldots,4$.

Definitions 9. Let an RTCP-net $\mathcal{N} = (\Sigma, P, T, A, C, G, I, E_M, E_S, M_0, S_0)$ be given. A marking *M* is said to be dead if the net $\mathcal{N}' = (\Sigma, P, T, A, C, G, I, E_M, E_S, M, S_0)$ is dead. A state *(M, S)* is said to be dead if the marking *M* is dead.

Live markings and states are defined analogously. A live net does not guarantee that each transition fires as often as the others. Some transitions may be *starved by* others.

Definition 10. Let an RTCP-net $\mathcal{N}$ and a firing sequence $\alpha \in \mathcal{L}(M_0,S_0)$ be given. A firing sequence α is said to be *fair* if it is either finite or infinite and each transition $t \in T$ appears infinitely often in α. The net $\mathcal{N}$ is said to be *fair* if every firing sequence $\alpha \in \mathcal{L}(M_0,S_0)$ is fair.

Definition 11. The *duration of* a firing sequence $\alpha = (t_1,b_1),(t_2,b_2),\ldots$ is the sum:

$$\delta(\alpha) = \sum_i \tau_i, \tag{6}$$

where values τ_i, $i = 1,2,\ldots,n-1$ denote passages of time between consecutive states (see

equation (5)).

Definition 12. Let (*M, S)* and (*M'*, *S'*) be the states of an RTCP-net $\mathcal{N}$ such that $(M', S') \in \mathcal{R}(M, S)$. A *time of transition from* the state (M, S) to (M', S'), denoted by $\delta((M, S), (M', S'))$, is the duration of any sequence a leading from the state (M, S) to (M', S').

The duration of a firing sequence is unambiguous, while a time of transition from one state to another is not. If there are a few firing sequences leading from the state *(M, S)* to *(M',S'),* we receive a few possibly different times of transition between these states. The most important ones are the minimal and maximal times of transition.

Analysis of RTCP-nets may be carried out using reachability graphs. The set of reachable states $\mathcal{R}(M_0, S_0)$ is represented as a weighted, directed graph. Each node corresponds to a unique state, consisting of a net marking and a time vector, such that the state is a result of firing of a transition. Each arc represents a change from a state (M_i, S_i) to a state (M_j, S_j) resulting from a passage of time $\tau \geq 0$ and a firing of a transition *t* in a binding $b \in \mathcal{B}(t)$.

Let's consider the net presented in Fig. 1. None transition is enabled in the state (M_2, S_2), but it is possible a passage of time $\tau = 6$ that leads to the state (M_2, S_2'). The transition *TurnOnSS* is enabled in the state and its firing leads to the state (M_3, S_3) where:

$$\begin{aligned} M_3 &= (ssOn, on, (on, on), off, active, on), \\ S_3 &= (0, 54, 0, -6, 2, 54). \end{aligned} \tag{7}$$

Thus, in the reachability graph, there will be nodes for the states (M_2, S_2) and (M_3, S_3), and an arc going from (M_2, S_2) to (M_3, S_3) with label $((TurnOnSS, ()), 6)$.

A finite reachability graph may be used to verify the RTCP-nets' properties presented in this section. Analysis of boundedness and conservativeness properties may be carried out by using markings of the graph nodes, while analysis of liveness and fairness properties may be carried out by using labels of arcs. Each label of an arc is a pair of a transition with its binding and a passage of time. The second element of a pair can be treated as the weight of the arc. Thus, arcs' weights capture the time taken by transition from one state to the next. (We consider only states that are results of transitions' firing). Using the reachability graph, one can find the minimal and maximal times of transition from one state to another. To do this we can use typical algorithms for finding the shortest or longest paths between two nodes in a directed graph (multigraph). However, a reachability graph for an RTCP-nets may be infinite even though the net is strongly bounded. In such a case it is not very useful for analysis purposes. More detail description of reachability graphs can be found in (Szpyrka 2006a).

One of the main advantages of strongly bounded RTCP-nets (in practical applications RTCP-nets are usually strongly bounded) is the possibility to present the set of reachable states of an RTCP-net using a finite coverability graph. Such a graph can be used to verify most of the RTCP-net's properties, including the timing ones.

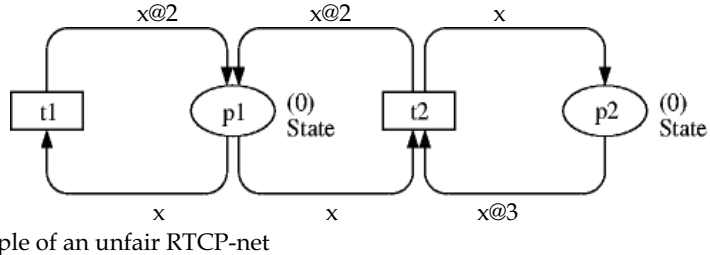


Fig. 2. Example of an unfair RTCP-net

Let's consider the RTCP-net presented in Fig. 2. The set Σ contains only one element `State = int with 0..1` and only one variable x is used. The initial marking $M_0 = (0,0)$ does not change while the net is working. The states change due to the changing of time stamps. The RTCP-net is not fair. The transition *t2* may be starved by the other one. Let's consider a firing sequence where only the transition *t1* is fired. In such a case the time stamp of the place *p2* will be infinitely decreasing. Therefore, the reachability graph for the considered net is infinite. A part of the reachability graph for the RTCP-net is shown in Fig. 3.

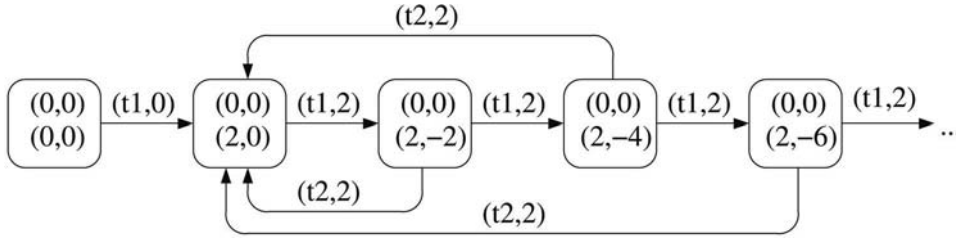


Fig. 3. Part of the reachability graph for the RTCP-net presented in Fig. 2

Let's consider two states of the RTCP-net (M_0, S_1) and (M_0, S_2), where $M_0 = (0,0)$, $S_1 = (2,-4)$ and $S_2 = (2,-6)$. The same transitions are enabled in both states and the same sequences of actions are feasible from the states. Both states have the same markings and the same *level of* tokens accessibility, i.e. we have to wait 2 time-units to take the token from the place *p1* and the token in the place *p*2 is already accessible. The token in the place *p*2 is accessible if its age is at least 3 time-units, i.e. the value of the time stamp is equal to or less than -3. It makes no difference whether the time stamp is equal to -4, -6, etc. The states (M_0, S_1) and (M_0, S_2) will be said to cover each other and only one node in the coverability graph will be used to represent them.

Definition 13. Let $p \in P$ be a place of an RTCP-net $\mathcal{N}$ and let $Out_A(p)$ denote the set of output arcs of the place *p*. *The maximal accessibility age* of the place *p* is the number:

$$\delta_{max}(p) = \max_{a \in Out_A(p)} \left\{ \max_{b \in \mathcal{B}(T(a))} E_S(a)_b \right\}. \tag{8}$$

The maximal accessibility age of a place *p* denotes the age when tokens in the place become accessible for all output transitions of the place.

Definition 14. Let $\mathcal{N}$ be an RTCP-net and let (M_1, S_1) and (M_2, S_2) be states of the net. The state (M_1, S_1) is said to cover the state (M_2, S_2) ($(M_1, S_1) \simeq (M_2, S_2)$) iff $M_1 = M_2$ and the following condition holds:

$$\forall p \in P: (S_1(p) = S_2(p)) \vee (S_1(p) \leq -\delta_{max}(p) \wedge S_2(p) \leq -\delta_{max}(p)). \tag{9}$$

Proposition 1. The *coverability relation* $\simeq$ is an equivalence relation on $\mathcal{R}(M_0, S_0)$.

Proposition 2. Let (M_1, S_1) and (M_2, S_2) be the states of an RTCP-net $\mathcal{N}$ such that $(M_1, S_1) \simeq (M_2, S_2)$. If $(M_1, S_1) \xrightarrow{(t,b)} (M_1', S_1')$ and $(M_2, S_2) \xrightarrow{(t,b)} (M_2', S_2')$, then $(M_1', S_1') \simeq (M_2', S_2')$.

Proposition 3. Let (M_1, S_1) and (M_2, S_2) be the states of an RTCP-net $\mathcal{N}$ such that $(M_1, S_1) \simeq (M_2, S_2)$. The following equality holds: $\mathcal{L}(M_1, S_1) = \mathcal{L}(M_2, S_2)$.

The reachability and coverability graphs are constructed in a similar way. They differ only

about the way a new node is added to the graph. For the coverability graph, after calculating a new node, we check first whether there already exists a node that covers the new one. If so, we add only a new arc that goes to the found state and the new one is omitted. Otherwise, the new state is added to the coverability graph together with the corresponding arc. The coverability graph contains only one node for each equivalence class of the coverability relation.
Let's consider coverability graph for the net presented in Fig. 2. After calculating the state (M_0, S_2) we affirm that there already exists the state (M_0, S_1) that covers it. Therefore, we add only an arc that goes back to the state (M_0, S_1). The coverability graph for the RTCP-net is shown in Fig. 4. The coverability graph for the net presented in Fig. 1 is shown in Fig. 5.

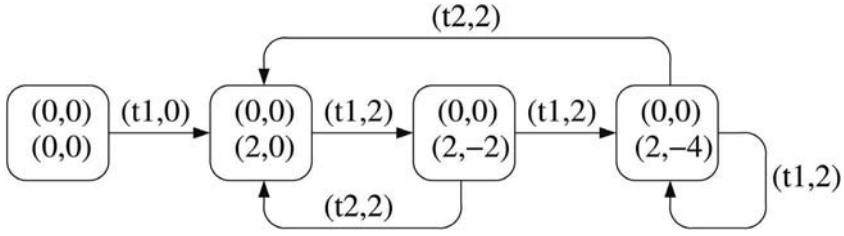


Fig. 4. Coverability graph for the RTCP-net presented in Fig. 2

Proposition 4. If an RTCP-net $\mathcal{N}$ is strongly bounded and each type $\Sigma_i \in \Sigma$ is finite, then the coverability graph is also finite.
Proofs for the presented propositions can be found in (Szpyrka 2006a).
The coverability graph for an RTCP-net provides similar capabilities of analysis of the net properties as the full reachability graph. It contains all reachable markings so it is possible to check the boundedness properties. The coverability graph contains similar arcs' labels as the reachability one (with the same pairs *(t,b))*, therefore, it is also possible to check the liveness properties. Possibilities of analysis of timing properties using coverability graphs are limited insignificantly so some states are not presented directly. To find the minimal and maximal times of the transition from one state to another we use the same algorithms as for reachability graphs. For more details see (Szpyrka 2006a).

4. Practical modelling with RTCP-nets

For the effective modelling RTCP-nets enable to distribute parts of the net across multiple subnets called pages. Hierarchical RTCP-nets are based on hierarchical CP-nets. Substitution transitions and fusion places (Jensen 1992-1997) are used to combine pages but they are a mere designing convenience. The former idea allows the user to refine a transition and its surrounding arcs to a more complex net, which usually gives a more precise and detailed description of the activity represented by the substitution transition. In comparison with CP-nets general ports are not allowed in RTCP-nets. Moreover, each socket node must have only one port node assigned and vice versa. Thus, a hierarchical net can be easily "squash" to a non-hierarchical one.
A fusion of places allows users to specify a set of places that should be considered as a single one. It means, that they all represent a single conceptual place, but are drawn as separate individual places (e.g. for clarity reasons). The places participating in such a fusion set may belong to several different pages. They must have the same types and initial

markings. Global fusion sets only are allowed in RTCP-nets.

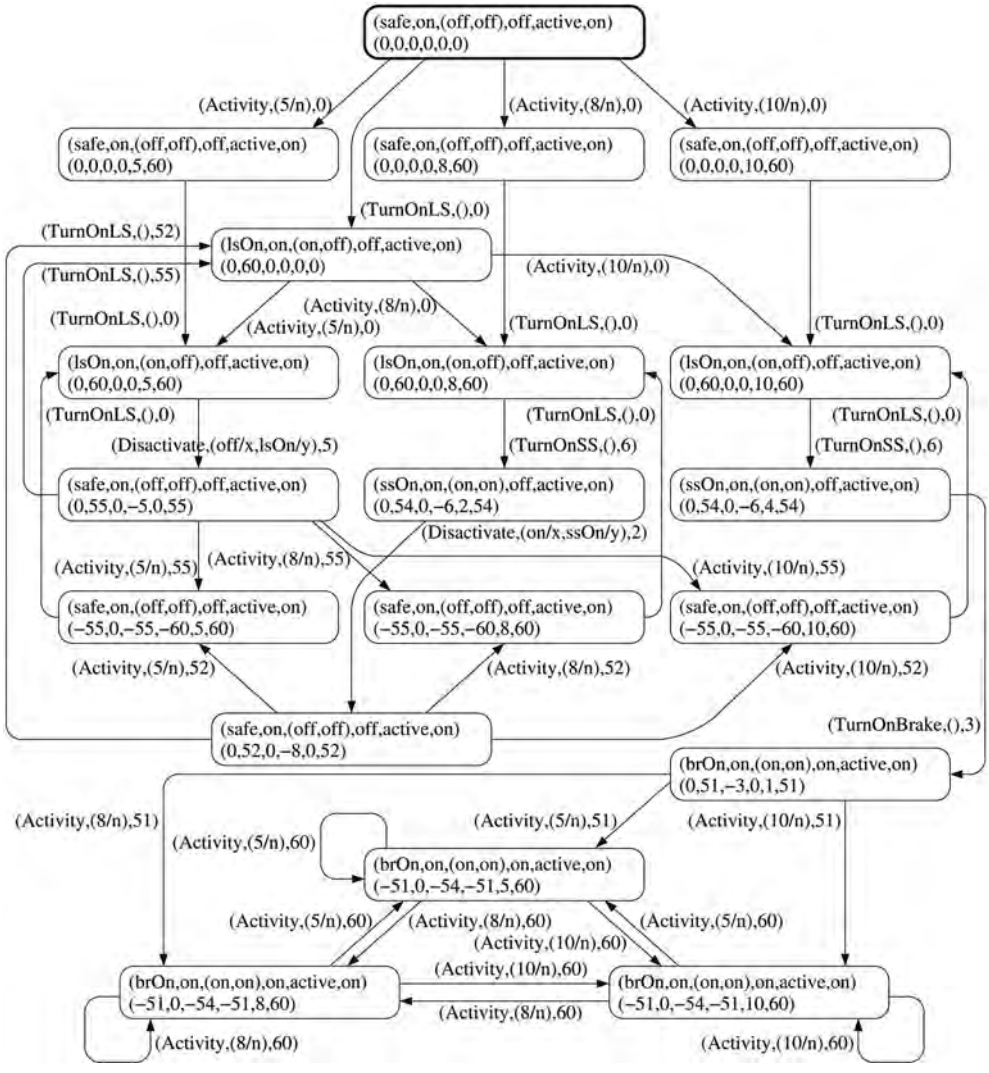


Fig. 5. Coverability graph for the RTCP-net presented in Fig. 1

4.1 Canonical form

A special form of hierarchical RTCP-nets called *canonical form* has been defined to speed up and facilitate drawing of models (Szpyrka and Szmuc 2006c). RTCP-nets in canonical form consist of four types of subnets with precisely defined structures: primary place pages, primary transition pages, linking pages, and D-nets. Such a model describes the structure of the corresponding system as well as its behaviour and functional aspects. Furthermore,

rule-based systems can be simply included into such models. The general structure of an RTCP-net in canonical form is shown in Fig. 6.

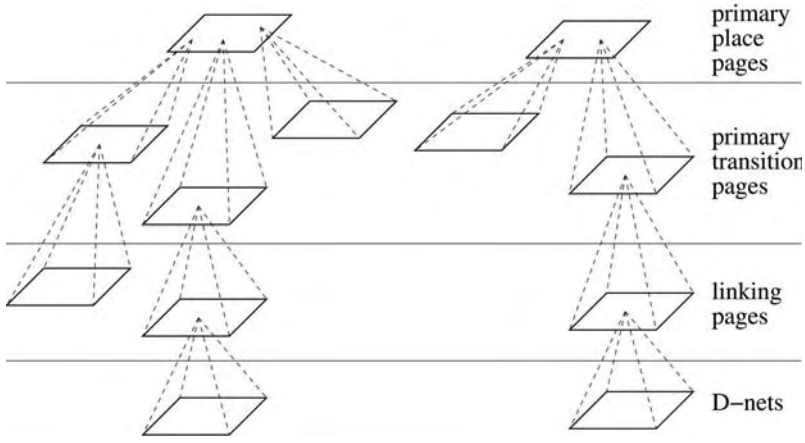


Fig. 6. General structure of an RTCP-net in canonical form

Moreover, it is assumed that an RTCP-net in canonical form satisfies some extra conditions. The set of places P is divided into two subsets: P_M, the set of *main places* and P_A, the set of *auxiliary places.* Main places represent the distinguished parts (elements) of a modelled system, e.g. objects. The set T of all transitions is also divided into two subsets: T_M *(main transitions)* and T_A *(auxiliary transitions).* Main transitions represent actions of a modelled system. Auxiliary places and transitions are used on subpages, which describe system activities in detail. Main places may be connected to main transitions only. Initial time stamps of auxiliary places must be equal to or less than 0. Moreover, if an arc goes from or to an auxiliary place, its time expression must be equal to 0.

Primary place pages are used to represent active objects (i.e. objects performing activities) and their activities. They are oriented towards objects presentation and are top level pages. Such a page is composed of one main place that represents the object and one main transition for each object activity. *Primary transition pages* are oriented towards activities' presentation and are second level pages. Such a page contains all the places, the values of which are necessary to execute the activity, i.e. the page is composed of one main transition that represents the activity and a few main places.

Linking pages belong to the functional level of a model. They are used (if necessary) to represent an algorithm that describes an activity in details. Moreover, a linking page is used as an interface for gluing the corresponding D-net into a model. Such a page is used to gather all necessary information for the D-net and to distribute the results of the D-net activity. A linking page contains port nodes for socket nodes from the corresponding primary transition page. The substitution transition (from the corresponding primary transition page) is split into two main transitions an input and an output one. All elements placed between those transitions are auxiliary ones, so there is no delay between firing of the input and output transitions. Hence, if time properties are considered, we can focus on primary transition pages and pass over their subpages. Any activity of a linking page starts with the firing of the input transition and ends with the firing of the output one. In addition, each occurrence of the input

transition must be followed by a sequence of transitions' occurrences such that the last of them is the output transition, and all the others are auxiliary ones. Any such activity is similar to a procedure call in programming languages.

D-nets (Szpyrka & Szmuc 2006a) are used to represent rule-based systems in a Petri net form. They are utilized to verify a rule-based system properties and constitute parts of an RTCP-net model. A D-net contains two places: a *conditional* and a *decision place.* Each decision rule is represented by a transition and its input and output arcs. A token placed in the conditional place denotes a sequence of values of conditional attributes. Similarly, a token placed in the decision place denotes a sequence of values of decision attributes. D-nets belong to the bottom level of the model. All its nodes belong to auxiliary ones. A simplified structure of these four types of pages is shown in Fig. 7.

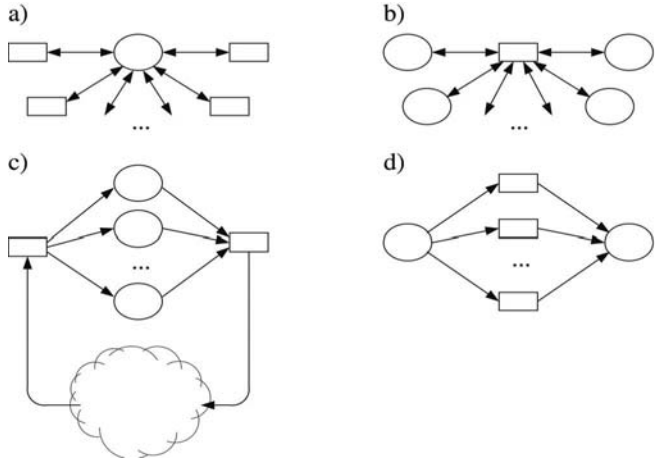


Fig. 7. Simplified structure of RTCP-net pages: a) primary place page; b) primary transition page; c) linking page; d) D-net

All connections among pages are presented using a page hierarchy graph. A node in such a graph represents a single page, and an arc represents a connection between a subpage and its substitution transition.

System decomposition is the first step of a model development. It starts with distinguishing objects that constitute the system. Objects are divided into *active,* i.e., objects performing activities, and *passive ones,* that do not perform any individual activity. An object is represented by a main place. For each object, a list of attributes and their types are defined. The Cartesian product of the defined types specifies the corresponding place type. Construction of primary place pages for active objects ends this development stage.

The next stage deals with description of model dynamic that is especially important for reactive systems. Transitions placed in primary place pages are usually substitution transitions. For each of these substitution transitions a primary transition page is drawn. Designing of a primary transition page is similar to declaring a procedure in Ada programming language. It is necessary to describe input, output and input/output parameters. If a primary transition page does not contain a substitution transition, then it constitutes a complete definition of the corresponding activity. After completion of this stage, RTCP-net represents all elements (objects) that constitute the modelled system and all

its activities.
The last stage is related to development of functional aspects of the system. Linking pages and D-nets (if necessary) are used for this purpose.

4.2 Railway traffic management system – case study

RTCP-nets can be used as modelling language for real embedded systems. A model of railway traffic management system for a real train station is discussed in this subsection. The system is used to ensure safe riding of trains through the station. It collects some information about current railway traffic and uses a rule-based system to choose routes for trains. The presented approach based on RTCP-nets seems to be valuable and worth consideration as an alternative for other approaches such as SDL language (Bacherini et al. 2003), statecharts (Banci et al. 2004) and others.
The size of a train station has a great influence on the size of the corresponding RTCPnet model. To give a brief outline of the presented approach a small train station (Czarna Tarnowska) has been chosen. The station belongs to the Polish railway line no 91 from Kraków to Medyka. This example seems to be suitable for RTCP-nets presentation.

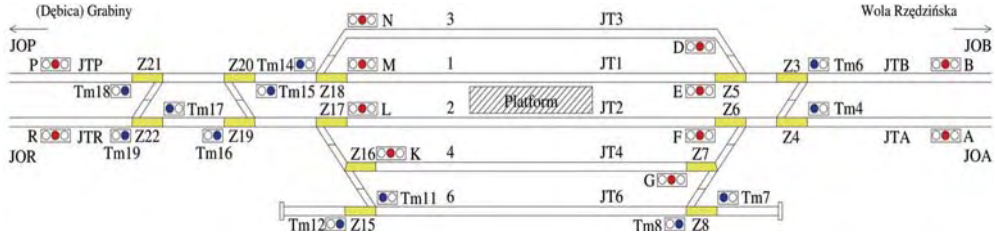


Fig. 8. Czarna Tarnowska – topology of the train station

The topology of the train station with original signs is shown in Fig. 8. The letters A, B, D, etc. stand for color light signals, the symbols Z3, Z4, Z5, etc. stand for turnouts and JTA, JTB, JT1, etc. stand for track segments. Some simplification have been introduced to reduced the size of the model. We are not interested in controlling local shunts so the track segment JT6 will not be considered. We assume that light signals display only two signals: *stop, way free*. Moreover, outside the station the trains can ride using the right track only.
A train can ride through the station only if a suitable route has been prepared for it i.e., suitable track segments must be free, we have to set turnouts and light signals and to guarantee exclusive rights to these elements for the train. Required position of turnouts for all possible routes are shown in Tab. 1. For example, the symbol B4 stands for the input route from the light signal B to the track no. 4. The symbol F2W stands for the output route from the track no. 2 (from the light signal F) to the right (to Wola Rzedzinska), etc. The route B4 can be used by a train only if: turnouts 7, 8, 15, 16 are closed, turnouts 3, 4, 6 are open, and the track segments JTB, JT4, JZ4/6 (a segment between turnouts 4 and 6), JZ7 (diagonal segment leading to the turnout 7) and JZ16 are free. The Tab. 2 shows which routes are mutually exclusive. The system is expected to choose suitable routes for moving trains. It should take under consideration that some trains should stop at the platform, while others are only moving through the station and two routes (an input and an output one) should be

prepared for them. In such a case, if it is not possible to prepare two routes, only an input one can be prepared.

	Turnouts								
Routes	3/4	5	6	7/8	15/16	17	18	19/20	21/22
B1	+	+							
B2	–		+	o+					
B3	+	–							
B4	–		–	+	o+				
R2				o+	o+	+		+	+
R4				o+	+	–		+	+
F2W	+		+	o+					
G2W	+		–	+					
K1D					+	–		–	+
L1D					o+	+		–	+
M1D							+	+	+
N1D							–	+	+

\+ closed turnout (the straight route);
– open turnout (the diverging route);
o+ closed turnout (from safety reasons).

Table 1. Required position of turnouts for all possible routes

	B1	B2	B3	B4	R2	R4	F2W	G2W	K1D	L1D	M1D	N1D
B1	–	x	x	x								xx
B2	x	–	x	x	xx		x	x				
B3	x	x	–	x								
B4	x	x	x	–	xx	xx	x	x				
R2		xx		xx	–	x		xx	x	x		
R4				xx	x	–			x	x		
F2W		x		x			–	x				
G2W		x		x	xx		x	–				
K1D					x	x			–	x	x	x
L1D					x	x			x	–	x	x
M1D									x	x	–	x
N1D	xx								x	x	x	–

x mutually exclusive (different position of turnouts);
xx mutually exclusive (safety reasons).

Table 2. Relationships between routs

The main part of the developed system is a rule-based system that is used to determine which routes should be prepared depending on the data collected from sensors. In the considered approach generalized decision tables (tables with non-atomic values of attributes, (Szpyrka & Szmuc 2006a)) are used to represent rule-based systems. A cell in such a decision table contains a formula that evaluates to a boolean value for conditional attributes, and to a single value (that belongs to the corresponding domain) for decision attributes. After verification such a decision table is transformed into a Petri nets form called D-net (Szpyrka & Szmuc 2006a).

The decision table for the considered model contains 20 conditional and 2 decision

attributes. The conditional attributes stand for: information about current position of the train (attribute JT) - before the light signal B, F, G, etc.; information about type of the train (attribute TT) - only moves through the station (1) or must stop at the platform (0); information about current status of track segments (attributes JT1, JT2, JT3, JT4, JOA, JOP) - a segment is free (0) or is taken (1); information about already prepared routes (attributes B1, B2, B3, etc.) - a route is already set (1) or not (0). The decision attributes In and Out represent the input and output routes (that will be prepared for the train) respectively. Domains for these attributes are defined as follows:

$D_{\mathrm{JT}} = \mathbf{with}\ \mathrm{b} \mid \mathrm{f} \mid \mathrm{g} \mid \mathrm{k} \mid \mathrm{l} \mid \mathrm{m} \mid \mathrm{n} \mid \mathrm{r},$
$D_{\mathrm{TT}} = \mathbf{int\ with}\ 1..2,$
$D_{\mathrm{JT1}} = D_{\mathrm{JT2}} = D_{\mathrm{JT3}} = D_{\mathrm{JT4}} = D_{\mathrm{JOA}} = D_{\mathrm{JOP}} = \mathbf{int\ with}\ 0..2,$
$D_{\mathrm{B1}} = D_{\mathrm{B2}} = \ldots = D_{\mathrm{N1D}} = \mathbf{int\ with}\ 0..1,$
$D_{\mathrm{In}} = \mathbf{with}\ \mathrm{b1} \mid \mathrm{b2} \mid \mathrm{b3} \mid \mathrm{b4} \mid \mathrm{r2} \mid \mathrm{r4} \mid \mathrm{none},$
$D_{\mathrm{Out}} = \mathbf{with}\ \mathrm{f2w} \mid \mathrm{g2w} \mid \mathrm{k1d} \mid \mathrm{l1d} \mid \mathrm{m1d} \mid \mathrm{n1d} \mid \mathrm{none}.$

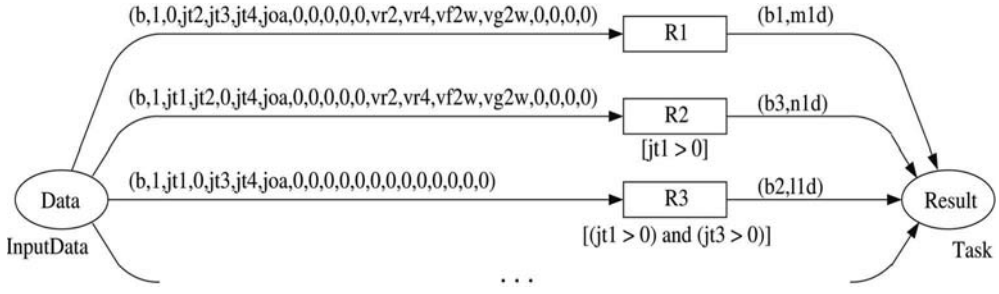


Fig. 9. Part of the D-net for the decision table presented in Table 3 and 4

The decision table is shown in Tab. 3 and 4. Moreover, the table contains 81 so-called negative rules (Szpyrka & Szmuc 2006a) that state in an explicit way that the particular combinations of values of conditional attributes are impossible or not allowed. The negative rules are used to check whether the table is complete and are usually omitted when the corresponding D-net is generated. Thus they will not be considered in the paper. Together with the negative rules, the rule-based system is complete and deterministic. A small part of the D-net for the decision table (only three rules) is shown in Fig. 9. Names beginning with "v" (e.g. vf2w) denote variables.

In the considered model two active objects are distinguished: *Driver* used to handle with trains requests and *SignalBox* used to set and unset routes. The first object contains information about all requests waiting for service, while the second one about states of all routes. Definitions of main types and variables used in the model are as follows:

```
color State = int with 0..1;
color TrainType = int with 0..2;
```

	JT	TT	JT1	JT2	JT3	JT4	JOA	JOP	B1	B2	B3
R1	JT = b	TT = 1	JT1 = 0	JT2	JT3	JT4	JOA	JOP = 0	B1 = 0	B2 = 0	B3 = 0
R2	JT = b	TT = 1	JT1 > 0	JT2	JT3 = 0	JT4	JOA	JOP = 0	B1 = 0	B2 = 0	B3 = 0
R3	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP = 0	B1 = 0	B2 = 0	B3 = 0
R4	JT = b	TT = 1	JT1 > 0	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP = 0	B1 = 0	B2 = 0	B3 = 0
R5	JT = r	TT = 1	JT1	JT2 = 0	JT3	JT4	JOA = 0	JOP	B1	B2 = 0	B3
R6	JT = r	TT = 1	JT1	JT2 > 0	JT3	JT4 = 0	JOA = 0	JOP	B1	B2 = 0	B3
R7	JT = b	TT = 2	JT1 = 0	JT2	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R8	JT = b	TT = 2	JT1 > 0	JT2 = 0	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R9	JT = b	TT = 2	JT1	JT2 = 0	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R10	JT = r	TT = 2	JT1	JT2 = 0	JT3	JT4	JOA	JOP	B1	B2 = 0	B3
R11	JT = b	TT = 1	JT1 = 0	JT2	JT3	JT4	JOA	JOP > 0	B1 = 0	B2 = 0	B3 = 0
R12	JT = b	TT = 1	JT1 = 0	JT2	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R13	JT = b	TT = 1	JT1 = 0	JT2	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R14	JT = b	TT = 1	JT1 = 0	JT2	JT3	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R15	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP > 0	B1 = 0	B2 = 0	B3 = 0
R16	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R17	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R18	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R19	JT = b	TT = 1	JT1 > 0	JT2 = 0	JT3 > 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R20	JT = b	TT = 1	JT1	JT2 = 0	JT3 > 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R21	JT = b	TT = 1	JT1 > 0	JT2	JT3 = 0	JT4	JOA	JOP > 0	B1 = 0	B2 = 0	B3 = 0
R22	JT = b	TT = 1	JT1 > 0	JT2	JT3 = 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R23	JT = b	TT = 1	JT1 > 0	JT2	JT3 = 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R24	JT = b	TT = 1	JT1 > 0	JT2	JT3 = 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R25	JT = b	TT = 1	JT1	JT2	JT3 = 0	JT4	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R26	JT = b	TT = 1	JT1 > 0	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP > 0	B1 = 0	B2 = 0	B3 = 0
R27	JT = b	TT = 1	JT1 > 0	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R28	JT = b	TT = 1	JT1 > 0	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R29	JT = b	TT = 1	JT1 > 0	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R30	JT = b	TT = 1	JT1	JT2 > 0	JT3 > 0	JT4 = 0	JOA	JOP	B1 = 0	B2 = 0	B3 = 0
R31	JT = r	TT = 1	JT1	JT2 = 0	JT3	JT4	JOA > 0	JOP	B1	B2 = 0	B3
R32	JT = r	TT = 1	JT1	JT2 = 0	JT3	JT4	JOA	JOP	B1	B2 = 0	B3
R33	JT = r	TT = 1	JT1	JT2 > 0	JT3	JT4 = 0	JOA > 0	JOP	B1	B2	B3
R34	JT = r	TT = 1	JT1	JT2	JT3	JT4 = 0	JOA	JOP	B1 = 0	B2 = 1	B3 = 0
R35	JT = r	TT = 1	JT1	JT2 > 0	JT3	JT4 = 0	JOA	JOP	B1	B2	B3
R36	JT = r	TT = 1	JT1	JT2	JT3	JT4 = 0	JOA	JOP	B1	B2	B3
R37	JT = k	TT = 1	JT1	JT2	JT3	JT4 > 0	JOA	JOP = 0	B1	B2	B3
R38	JT = l	TT	JT1	JT2 > 0	JT3	JT4	JOA	JOP = 0	B1	B2	B3
R39	JT = m	TT	JT1 > 0	JT2	JT3	JT4	JOA	JOP = 0	B1	B2	B3
R40	JT = n	TT = 1	JT1	JT2	JT3 > 0	JT4	JOA	JOP = 0	B1 = 0	B2	B3
R41	JT = f	TT	JT1	JT2 > 0	JT3	JT4	JOA = 0	JOP	B1	B2 = 0	B3
R42	JT = g	TT = 1	JT1	JT2	JT3	JT4 > 0	JOA = 0	JOP	B1	B2 = 0	B3

Table 3. Decision table (part 1)

```
color RoutesStates = product State * State *... (State^12);
color Route = with b1 | b2 | b3 | ... | n1d | none;
color Task = product Route * Route;
color LightSignal = with a | b | d | e | f | ...;
color Request = product LightSignal * TrainType;
var x1, x2, x3,..., y1, y2, y3,... : State;
var z1, z2 : Route;
var s1 : LightSignal;
var p1, p2, p3, p4, p5, p6, p7 : TrainType;
```

B4	R2	R4	F2W	G2W	K1D	L1D	M1D	N1D	In	Out
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	b1	m1d
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	b3	n1d
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 0	N1D = 0	b2	l1d
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 0	N1D = 0	b4	k1d
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r2	f2w
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r4	g2w
B4 = 0	R2	R4	F2W	G2W	K1D	L1D	M1D	N1D = 0	b1	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D	L1D	M1D	N1D	b2	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 0	N1D = 1	b2	none
B4 = 0	R2 = 0	R4 = 0	F2W	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r2	none
B4 = 0	R2	R4	F2W	G2W	K1D	L1D	M1D	N1D = 0	b1	none
B4 = 0	R2	R4	F2W	G2W	K1D = 1	L1D = 0	M1D = 0	N1D = 0	b1	none
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 1	M1D = 0	N1D = 0	b1	none
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 1	N1D = 0	b1	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D	L1D	M1D	N1D	b2	none
B4 = 0	R2 = 0	R4 = 1	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	b2	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D = 1	L1D = 0	M1D = 0	N1D = 0	b2	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D = 0	L1D = 1	M1D = 0	N1D = 0	b2	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 1	N1D = 0	b2	none
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 0	N1D = 1	b2	none
B4 = 0	R2	R4	F2W	G2W	K1D	L1D	M1D	N1D	b3	none
B4 = 0	R2	R4	F2W	G2W	K1D = 1	L1D = 0	M1D = 0	N1D = 0	b3	none
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 1	M1D = 0	N1D = 0	b3	none
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 1	N1D = 0	b3	none
B4 = 0	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 1	b3	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D	L1D	M1D	N1D	b4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 1	L1D = 0	M1D = 0	N1D = 0	b4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 1	M1D = 0	N1D = 0	b4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 1	N1D = 0	b4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 0	K1D = 0	L1D = 0	M1D = 0	N1D = 1	b4	none
B4 = 0	R2 = 0	R4 = 0	F2W	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r2	none
B4 = 0	R2 = 0	R4 = 0	F2W = 1	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r2	none
B4 = 0	R2 = 0	R4 = 0	F2W	G2W	K1D = 0	L1D = 0	M1D	N1D	r4	none
B4 = 0	R2 = 0	R4 = 0	F2W	G2W	K1D = 0	L1D = 0	M1D	N1D	r4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 1	G2W = 0	K1D = 0	L1D = 0	M1D	N1D	r4	none
B4 = 0	R2 = 0	R4 = 0	F2W = 0	G2W = 1	K1D = 0	L1D = 0	M1D	N1D	r4	none
B4	R2 = 0	R4 = 0	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	none	k1d
B4	R2 = 0	R4 = 0	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	none	l1d
B4	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	none	m1d
B4	R2	R4	F2W	G2W	K1D = 0	L1D = 0	M1D = 0	N1D = 0	none	n1d
B4 = 0	R2	R4	F2W = 0	G2W = 0	K1D	L1D	M1D	N1D	none	f2w
B4 = 0	R2 = 0	R4	F2W = 0	G2W = 0	K1D	L1D	M1D	N1D	none	g2w

Table 4. Decision table (part 2)

As it was said before, primary place pages are used to represent active objects. Such pages for objects *Driver* and *SignalBox* are shown in Fig. 10 and 11 respectively.

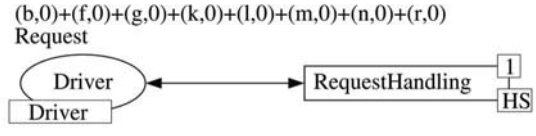


Fig. 10. Primary place page *Driver*

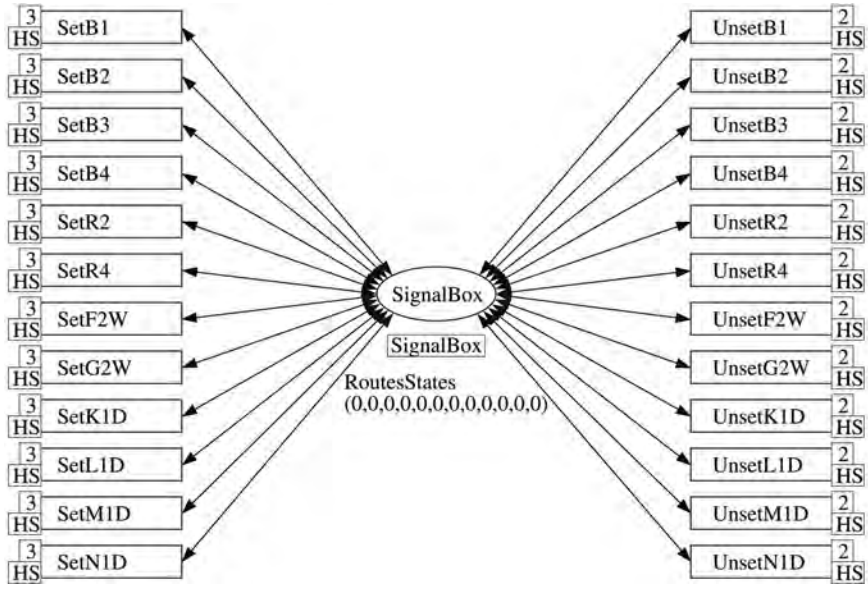


Fig. 11. Primary place page *SignalBox*

The RTCP-net model contains some passive objects such as light signals, turnouts and track segments. For simplicity turnouts that work together e.g. Z34 and Z78 are considered as single objects. Places that represent light signals and turnouts have the colour State assigned. The value 0 denotes that the stop signal is displayed or that the turnout is closed. Places that represent track segments have the colour TrainType assigned. A positive value denotes that the track segment is taken while the value 2 denotes that the train that occupies it must stop at the platform.

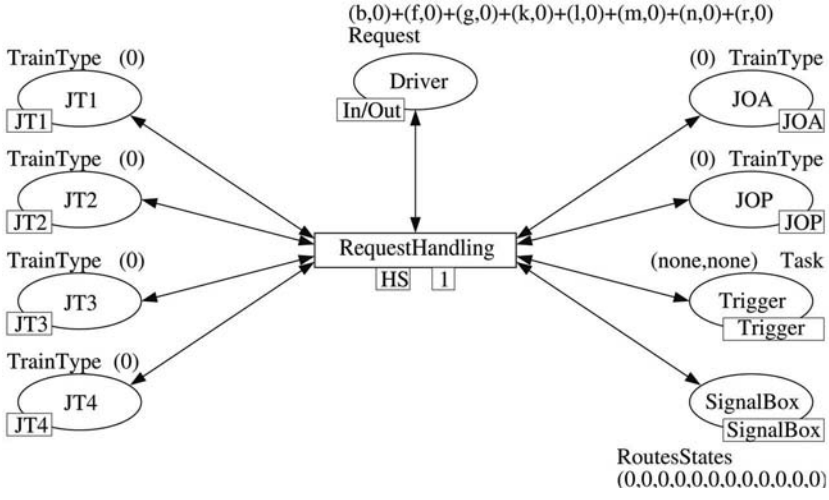


Fig. 12. Primary transition page *RequestHandling*

Transitions placed in primary place pages are usually substitution transitions. For each of them a *primary transition page* is drawn. Primary transition page for the *RequestHandling* is shown in Fig. 12. The transition performs on basis of the considered D-net. To connect the D-net with the substitution transition *RequestHandling* from the Fig. 12 a linking page must be used. The linking page used to gather all necessary information for the D-net and to distribute the results of the D-net activity is shown in Fig. 13.

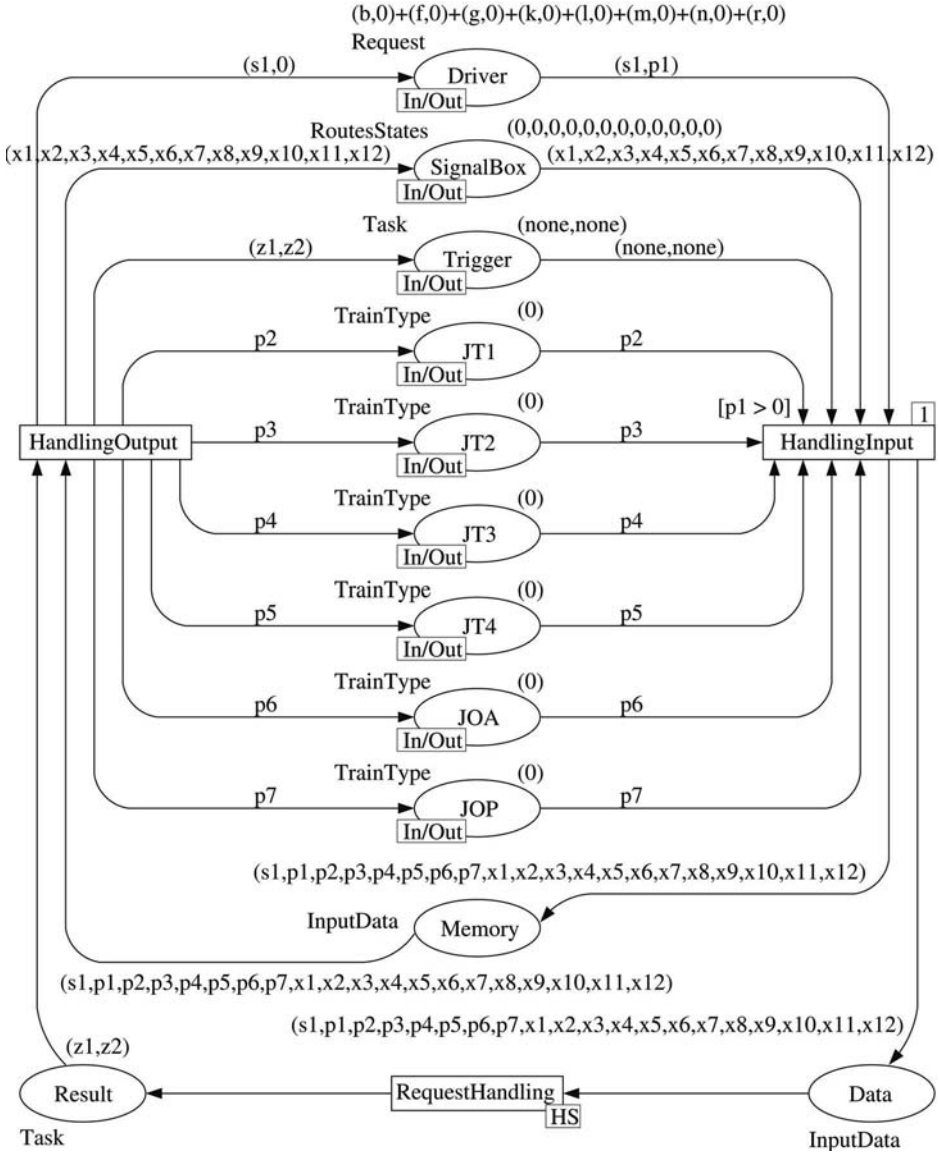


Fig. 13. Linking page for the transition

Primary transition pages for transitions *SetB1* and *UnsetK1D* are shown in Fig. 14 and 15 respectively. A route is unset if the second of the light signals (set for the route) displays the stop signal (light signals switch to stop signal when trains move) and the corresponding track segment is free. As the result of unset operation all used turnouts are switch to the closed position. Pages for other *Set* and *Unset* transitions are designed in similar way.

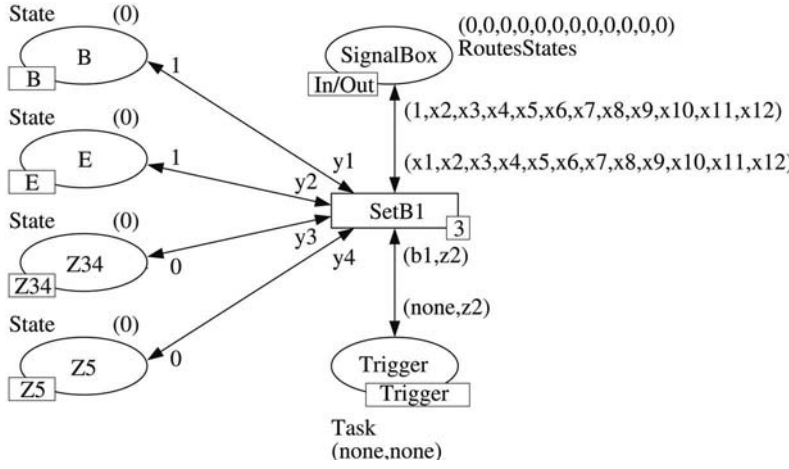


Fig. 14. Primary transition page *SetB1*

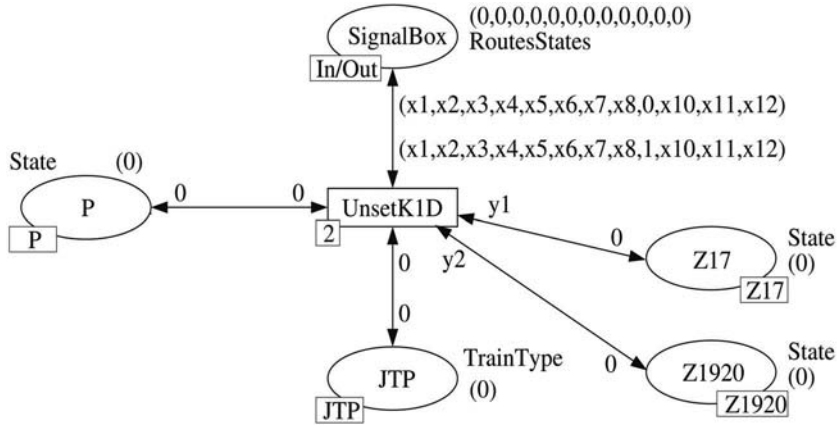


Fig. 15. Primary transition page *UnsetK1D*

A part of the page hierarchy graph for the RTCP-net is shown in Fig. 16. The complete model contains also pages that represent trains and are use to simulate the train traffic.

Verification of such a model is carried out in two ways. In the first one, simulation is used to check how the model works. Simulation of an RTCP-net model is similar to a program debugging. It can be used to check whether the model performs as expected or for some statistical analysis of its properties. A small part of a simulation report for the considered

model is presented below. Each part of the report contains the following pieces of information: a state number, markings of places, time stamps of places (in square brackets), a transition name, a binding of the transition and a passage of time (before the transition can fire). For example in the state 13 there is possible a passage of time equal to 20 time-units and then the transition *FreeFA1* (connected with a train moving) in the binding *b = (1/p1)* is fired.

```
13:
Driver   : [0] (b,0)+(f,0)+(g,0)+(k,0)+(l,0)+(m,0)+(n,0)+(r,2)
SignalBox: [0] (0,0,0,0,0,0,1,0,0,0,0,0)
Trigger  : [0] (none,none)
A, B, D, E, F, G, K, L, M, N, P, R
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
[0. -80. -80. -80. 0. -80. -80. -60. -80. -80. -80. -80]

JOA, JOB, JOP, JOR, JTA, JTB, JTP, JTR, JT1, JT2, JT3, JT4
1,   0,   0,   2,   1,   0,   0,   0,   0,   1,   0,   0
[0, -20, 0, -20, 0, -80, -80, -60, 0, 0, 0, 0]
Z34, Z5, Z6, Z78, Z1516, Z17, Z18, Z1920, Z2122
0,   0,  0,  0,   0,     0,   0,   0,     0,
[0, -80, 0, 0, -80, -80, -80, -80, -80]
Trains:  DW1,   DW2,   DW3,   WD1,   WD2,   WD3,   WD4,   WD5
         (w,f), (r,x), (x,x), (x,x), (x,x), (x,x), (x,x), (x,x)
[20, -20, -80, -20, -80, -80, -80, -80]
TimerDW, TimerWD
(0)+(1), (0)+(1)
[40, 40]
------------------------------------------------------------------------
--((FreeFA1.(1/p1)), 20)--> 14
```

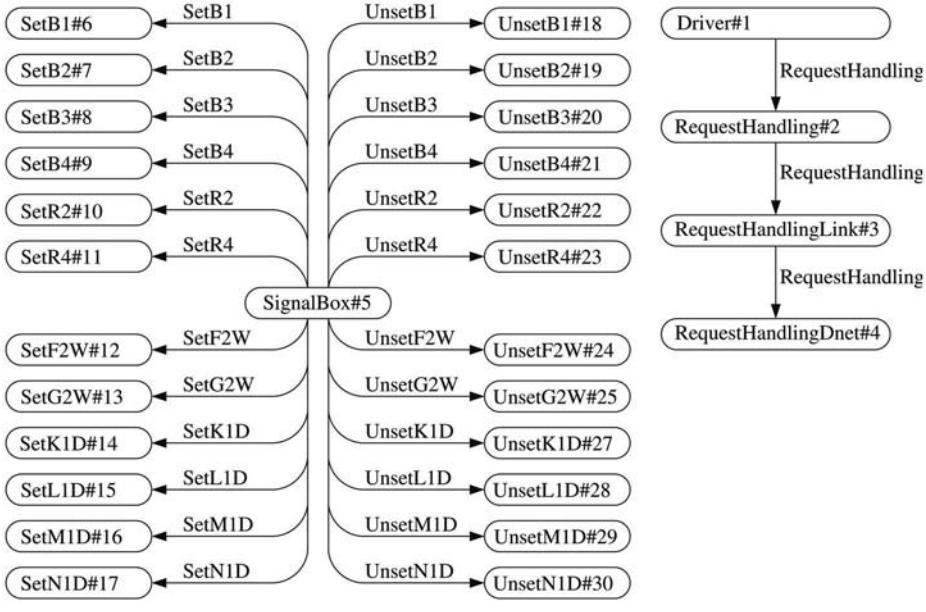


Fig. 16. Part of the page hierarchy graph

The real formal verification is based on coverability graphs. The textual form of the coverability graph is similar to the one of simulation report. In comparison with the simulation report, the coverability graph represents not one but all possible paths of the system performance. The following conclusions are results of the coverability graph analysis:

- The net is strongly bounded, live but it is not fair.
- Mutually exclusive routes are never set at the same time.
- Routes are set only if there is a need to do so, and are unset immediately after the corresponding train leaves the suitable track segments.
- The *way free* signal is displayed only if a suitable route is set and the *stop* signal is set only as a result of trains mowing.
- Trains never ride through taken track segments.

It should be also emphasized that the verification of the decision table (at its design stage) has also great influence on the presented model properties. The complete and deterministic decision table guarantees correct performance of the *RequestHandling* transition.

The design and verification of the presented model has been done with a software support. CASE tools for RTCP-nets called *Adder Tools* are being developed at AGH University of Science and Technology in Krakow. *Adder Tools* contain:

- *Adder Designer*- for design and verification of rule-based systems;
- *Adder Editor*- for design of RTCP-nets;
- *Adder Simulator*- for simulation of RTCP-nets.

Some preliminary version of the software and more information about it can be found at *http://adder. ia. agh. edu.pl.*

5. Conclusions

RTCP-nets are an adaptation of CP-nets to make modelling and verification of embedded systems easier and more efficient. Based upon the experience with application of CP-nets for embedded systems' modelling, some modifications were introduced in order to make timed CP-nets more suitable for this purpose. The main advantage of the presented formalism is the new time model. Together with transitions' priorities, the time model enable designers direct modelling of task priorities, timeouts, etc. that are typical for concurrent programming.

The next advantage of RTCP-nets is the possibility of analysis of model properties with coverability graphs. Timed CP-nets can be also used to model embedded systems. A few different analysis methods have been proposed for untimed CP-nets but analysis of the timed ones may be difficult. In most cases, the state space of a live timed CP-net is infinite, so it is impossible to construct a full reachability graph that allows to analyse timing properties. To reduce such an infinite state space a few kinds of reduced reachability graphs have been defined, for example: graphs with stubborn sets, (Kristensen & Valmari 1998), graphs with equivalence classes (Jorgensen & Kristensen 1997), graphs with symmetries (Jorgensen & Kristensen 1999) and others. In most cases, analysis of time properties is impossible or limited significantly. In case of RTCP-nets coverability graphs can be used for these purposes. It has been proved (Szpyrka 2006a) that for strongly bounded RTCP-nets we may construct a finite coverability graph. Such a graph can be used for the analysis of typical Petri nets' properties as well as timing ones.

The other advantage of RTCP-nets is the way hierarchical models are constructed. Using of

the canonical form speeds up and facilitate drawing of models. Moreover, some parts of a model can be generated automatically, because they have precisely defined structure.
Furthermore, an RTCP-net in canonical form represents the structure of a modelled system, its dynamic and also functional relationships. Each part (object) of the modelled system and each system activity is represented by a distinguished part of the corresponding RTCP-net. Hence, it is possible to identify parts of Ada source code that should be defined e.g.: tasks, protected objects, suspending objects, etc. Some aspects of the transformation from the formal RTCP-net model into an implementation in Ada 2005 programming language can be found in (Szpyrka 2006b). The received source code meets the requirements of the Ravenscar profile (Burns et al. 2003). The transformation algorithm is the first step toward working out tools for automatic transformation of an RTCP-net into Ada source code framework. The algorithm has been successfully used for generation Ada 2005 source code for the railway traffic management system presented in the chapter.
Our future plans will focus on the development of Adder tools capabilities (e.g. wizards for automatic generation of some part of models) and implementation of analysis methods. Development of the software seems to be the most important task to put the presented theory into practice.

6. References

Bacherini, S.; Bianchi, S.; Capecchi, L; Becheri, C.; Felleca, A.; Fantechi, A. & Spinicci, E. (2003). Modelling a railway signalling system using SDL, *Proceedings of FORMS 2003 Symposium on Formal Methods for Railway Operation and Control Systems,* pp. 107-113, ISBN 963-9457-450, Budapest, Hungary, May 2003, L'Harmattan Hongrie, Budapest

Banci, M.; Fantechi, A. & Gnesi, S. (2004). The role of formal methods in developing a distributed railway interlocking system, *Proceedings of the 5th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004),* pp. 220-230, ISBN 3-9803363-8-7, Braunschweig, Germany, December 2004, Technical University of Braunschweig

Barnes, J. (2006). *Programming in Ada 2005,* Addison Wesley, ISBN 0-321-34078-7, Harlow, England

Burns, A.; Dobbing, B. & Vardanega, T. (2003). *Guide for the Use of the Ada Ravenscar Profile in High Integrity Systems,* Technical Report No. YCS-2003-348, University of York

Cheng, A. M. K. (2002). *Real-time Systems. Scheduling, Analysis, and Verification,* Wiley Interscience, ISBN 978-0-471-18406-5, New Jersey

Jensen, K. (1992-1997). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use,* Vol. 1-3, Springer-Verlag, ISBN 3-540-62867-3, Berlin

Jorgensen, J.B. & Kristensen, L.M. (1997). Verification of coloured Petri nets using state spaces with equivalence classes, *Proceedings of the Workshop on Petri Nets in System Engineering, Modelling, Verification and Validation,* pp. 20-31, ISBN 3-89586-597-4, Hamburg, Germany, September 1997, Universitat Hamburg

Jorgensen, J.B. & Kristensen, L.M. (1999). Computer aided verification of Lamport's fast mutual exclusion algorithm using coloured Petri nets occurrence graphs with symmetries. *IEEE Transactions on Parallel and Distributed Systems,* Vol. 10, No. 7 (1999) 714-732, ISSN 1045-9219

Kristensen, L.M. & Valmari, A. (1998). Finding stubborn sets of coloured Petri nets without

unfolding, *Proceedings of the 19th International Conference on Application and Theory of Petri Nets,* pp. 104-123, ISBN 3-540-64677-9, Lisbon, Portugal, June 1998, LNCS, Vol. 1420, Springer-Verlag, London

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE,* Vol. 77, No. 4, (April 1989) 541-580, ISSN 0018-9219

Sommerville, I. (2004). *Software Engineering,* Pearson Education Limited, ISBN 0-321-21026-3, Boston

Szpyrka, M. (2006a). Analysis of RTCP-nets with reachability graphs. *Fundamenta Informaticae,* Vol. 74, No. 2-3, (2006) 375-390, ISSN 0169-2968

Szpyrka, M. (2006b). Development of embedded systems - from RTCP-net model to Ada code. *Concurrency, Specification and Programming CS&P'2006,* Vol. 2, pp. 231-242, ISSN 0863-095X, Wandlitz, Germany, September 2006, Informatik Berichte, No. 206, Humboldt-Universitatzu Berlin

Szpyrka, M. & Szmuc, T. (2006a). D-nets- Petri net form of rule-based systems. *Foundations of Computing and Decision Sciences,* Vol. 31, No. 2 (2006) 157-167, ISSN 0867-6356

Szpyrka, M. & Szmuc, T. (2006b). Verification of automatic train protection systems with RTCP-nets. Proceedings of the 25th International Conference on Computer Safety, Security and Reliability, pp. 344-357, ISBN 3-540-45762-3, Gdansk, Poland, September 2006, LNCS, Vol. 4166, Springer-Verlag, Berlin

Szpyrka, M. & Szmuc, T. (2006c). Integrated approach to modelling and analysis using RTCP-nets. *IFIP International Federation for Information Processing,* Vol. 227, 115-120, ISBN 978-0-387-39387-2, Springer, New York

3

Petri Net Based Modelling of Communication in Systems on Chip

Holger Blume, Thorsten von Sydow, Jochen Schleifer and Tobias G. Noll
Chair of Electrical Engineering and Computer Systems
RWTH Aachen University
Germany

1. Motivation

Due to the progress of modern microelectronics the complexity of integrated electronic systems is steadily increasing. For example, the number of transistors which can be integrated on a single piece of silicon doubles every 18 months according to Moore's Law (Moore, 1965). At the same time, the costs for manufacturing deep-sub-µ devices with feature sizes down to 45 nm are dramatically increasing.
Due to this progress, today, complete systems are integrated on a single silicon die as so-called Systems on Chip (SoCs). The huge complexity of these SoCs and the very high manufacturing costs demand sophisticated design strategies as it is not possible to simulate a sufficiently large number of implementation alternatives in advance. Furthermore, errors within the design process lead to dramatically increased costs.
Therefore, the field of model based design space exploration (DSE) is of increasing importance. Model based DSE allows a reduction of the number of implementation alternatives in an early stage of the design process by quantitative analysis of possible implementation alternatives (Blume, 2005).
Especially, the design of a sophisticated communication structure on a SoC is of great interest. For SoCs with moderate complexity mainly bus based communication structures are applied, but this is not sufficient for modern high complex SoCs, since bus based communication provides only very limited scalability, reduced bandwidth and no guaranteed latencies. Furthermore, with a high number of system components the need for simultaneous communication between different communicating units increases. All these requirements are already known from multi computer networks. Therefore, for complex on-chip communication requirements also network-like structures are considered. Hence, the concept of multi computer networks is transferred and adapted to on-chip communication problems building so-called Networks on Chip (NoCs) featuring in future the communication infrastructure for many processor cores.
Generally, NoCs consist of

- network-interfaces (NI), where clients like e.g. processor cores can access the NoC,
- routing-switches (RS), which route the data through the NoC and
- links, through which the data is transported (see Fig. 1).

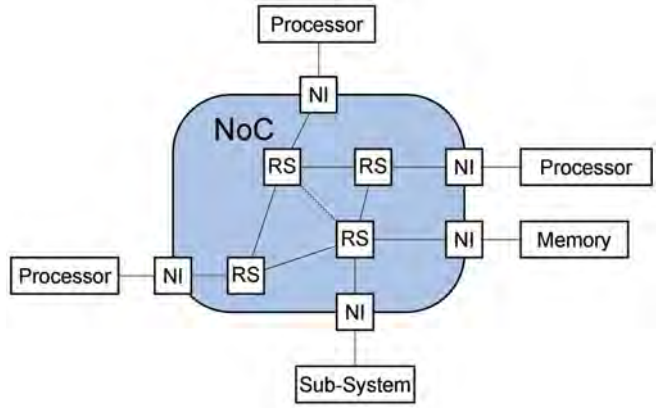


Fig. 1. Network-on-Chip (NI: Network-Interface, RS: Routing-Switch)

These NoCs imply a huge parameter space featuring parameters like network topology, routing strategy, link properties, arbitration mechanisms etc.. Some of these parameters are roughly sketched in the following:
Topology in this case is the way of connecting the various network components to each other, common examples (see Fig. 2) are networks based on mesh, torus and ring topologies (Bjerregaard, 2006), besides further regular topologies also heterogeneous and/or hierarchical topologies as well as completely irregular ones (for example optimized for specialized signal processing tasks) are discussed.

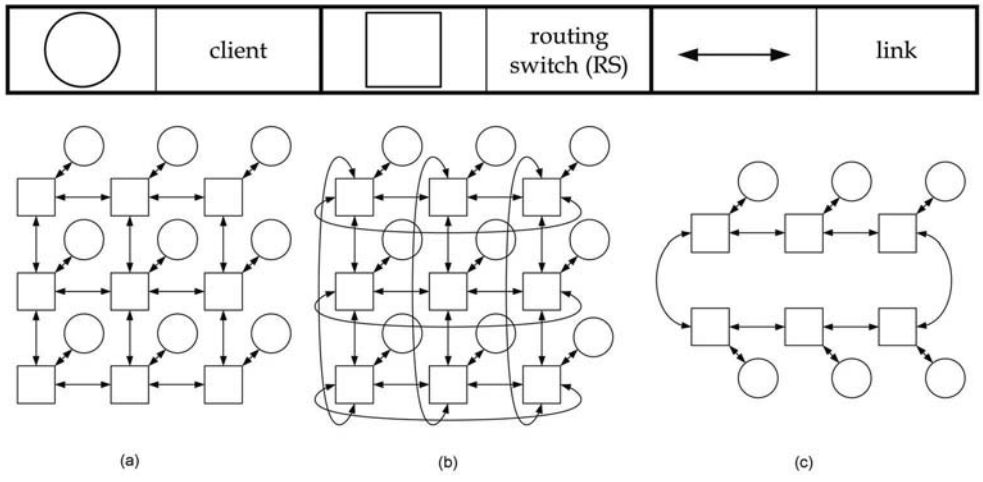


Fig. 2. Common NoC topologies: mesh (a), torus (b), ring (c)

The *switching concept* defines the way information is sent through the network. Concepts for this are line and packet switching. In a line switched network a complete route from source to destination is established before information is passed along this route. In the packet switching approach information is divided into small packets that are delivered

independently of each other. While line switching generates a considerable overhead for route establishment there is no need to send destination information along with each message and vice versa for packet switching. Furthermore, concepts such as wormhole routing combine characteristics of both approaches (Duato, 2003).
In any of the cases described before, the actual route through the network is determined by the *routing algorithm*. In each network node, information is routed from an input to an output port according to the routing algorithm. These algorithms can be divided into static and adaptive algorithms as well as minimal-path and non-minimal-path ones. When using a static routing algorithm there is only a single route for each possible pair of source and destination. Adaptive algorithms allow for different routes dependent on the current network state and generally tend to reduce congestion for the cost of higher complexity. Minimal-path routing algorithms only consider those routes with minimum possible length while non-minimal-path algorithms also regard further routes featuring non-minimal path lengths.
Arbitration mechanisms define the way to resolve resource conflicts; this can range from simple first-come-first-serve or Round Robin schemes to complex schemes including priority access and disruption of routes.
All of these parameters have a significant influence on congestion, latency and network load, thereby affecting and possibly limiting SoC and NoC performance.
It is a key task of modern SoC and NoC design to efficiently explore the design space regarding aspects like performance, flexibility and power consumption presumably in an early stage of the design flow in order to reduce design time and design costs.
Different approaches for exploring the design space concerning performance aspects have been proposed:

- Emulation on FPGA based platforms (Neuenhahn, 2006)
- Simulative approaches, e.g. applying SystemC (Kogel, 2003), (Madsen, 2004), (Sonntag, 2005)
- Combined simulative-analytic approaches (Lahiri, 2001)
- formal communication modelling and refinement systems applying dedicated modelling languages like the Action Systems Formalism (Plosila, 2004)
- stochastic approaches applying Markov Models (Mickle, 1998), Queuing Theory (Kleinrock, 1975) or different forms of Petri Nets incl. deterministic and stochastic Petri Nets (DSPN) (Ciardo, 1995), (Blume, 2006), (Blume, 2007) and Coloured Petri Nets (CPN) (Zaitsev, 2004)

Each of these techniques provides its individual advantages and disadvantages. For example, simulative approaches based on SystemC like (Kogel, 2003) provide highly accurate results but suffer from long simulation times, making them not appropriate for an early stage of communication modelling and evaluation. Emulation of communication architectures and scenarios on FPGA based platforms (Neuenhahn, 2006) provides on one hand the possibility to quickly acquire results for different aspects. If a suitable FPGA based model is available it is much faster to attain results than using a simulation based method. On the other hand the modelling and realization effort of the emulation (incl. the synthesis of the NoC on the FPGA) is very high. The complexity of the modelled scenarios is limited by the capacity of the used FPGA. Recently, communication modelling approaches which are based on so-called deterministic and stochastic Petri Nets (DSPNs) have been presented. In (Blume, 2006), (Blume, 2007) it could be shown that with the application of these DSPN

modelling techniques it is possible to efficiently trade modelling effort and modelling accuracy. Basic but exemplary test scenarios like resource conflicts in state-of-the-art DSP architectures, basic bus based communication test cases and basic NoC structures demonstrate a very good modelling accuracy at low modelling effort.
In this chapter the usability of different Petri Net based modelling techniques like DSPNs and CPNs for modelling complex NoC communication scenarios is investigated and their specific properties are discussed. This chapter is structured as follows: section 2 provides an introduction into the Petri Net variants DSPN and CPN. In section 3 these modelling techniques are applied in order to model different forms of on-chip communication. The corresponding accuracy of the models compared to values which were derived using FPGA and DSP based testbeds are provided and discussed. Furthermore, the related modelling effort is analyzed. Section 4 provides a conclusion and a short outlook to possible future applications of Petri Net based techniques in the domain of design space exploration for NoC-architectures.

2. Introduction to Petri nets

In the following section a short introduction to the Petri Net methods which have been applied in context of this chapter is given. The modelling with these specific methods will be illustrated by means of descriptive basic examples. First of all, the design with so called deterministic and stochastic Petri Nets (DSPNs) is sketched. Afterwards, coloured Petri Nets which extend the possibilities of DSPNs are briefly presented.

2.1 Deterministic and stochastic Petri nets

Deterministic and stochastic Petri Nets have been introduced in 1987 by Ajmone Marslan and Chiola (Ajmone Marslan, 1987) as an extension of classical Petri Nets. DSPNs extend the modelling possibilities of classical Petri Nets by introducing the concept of deterministic transition times. In the following, only a subset of all features provided by DSPNs is discussed. For a thorough overview see e.g. (Lindemann, 1998).
Petri Nets consist of so-called places, arcs and transitions. Places, depicted as circles in the graphical representation, correspond to states of e.g. system components. E.g. a place could be named *copy word* to illustrate that this place represents the state of copying a word. Places can be unmarked or marked with one or even more tokens. This illustrates that the corresponding place is currently allocated. E.g. if a place called *copy word* is marked, the associated component is in the state of copying a word.
In Petri Nets a state change is modelled by means of so called (timed) transitions. Three types are differentiated in DSPNs: immediate transitions, transitions with a probability density function of their delay (e.g.: negative exponential) or deterministic transitions with a fixed delay.
Transitions and places are connected via arcs. There are two types of arcs, regular or inhibitor arcs. Inhibitor arcs are identified by a small inversion circle instead of an arrowhead at the destination end of it (see Fig. 3). If more than one input place is connected to a transition via regular arcs, the transition will only be activated when all connected places are marked. In case of one or more of these arcs being inhibitor arcs the transition will not fire if the corresponding place is marked. Furthermore, a numeric weight can be

assigned to each arc. A weighted arc is only activated if the number of tokens, located in the place the arc is originating from, is greater or equal than the assigned weight.
The form of graphical representation is often used to build DSPNs. The underlying mathematical representation of DSPNs can be specified as a nine-tuple

$$DSPN = \left(P, T, I(\cdot), O(\cdot), H(\cdot), \Pi(\cdot), M_0, D(\cdot), W(\cdot)\right)$$

with

- P, a finite number of places,
- T, a finite number of transitions,
- $I(\cdot), O(\cdot), H(\cdot)$ denote the input-, output- and inhibitor-functions, which connect transitions and places,
- $\Pi(\cdot)$ denotes the firing-priority-function (specifying firing-priority-levels) for all immediate transitions,
- M_0 denotes the initial marking of the DSPNs,
- $D(\cdot)$ denotes the firing-delay-function (specifying the average delay) for timed transitions,
- $W(\cdot)$ denotes the firing-weight-function, which specifies the weights which are associated to each transition.

When a Petri Net model has been implemented by use of a graphical design tool or by directly defining the characteristic DSPN nine-tuple, the belonging mathematical models of the implemented Petri Net can be analyzed. Then, this analysis yields for example

- the static expectation value of the marking of places (occurrence of tokens at a given place),
- the stationary probability for the occurrence of a specific marking of a place,
- the average number of tokens passing a transition per unit of time, i.e. the throughput of a transition.

For the acquisition of results three different approaches exist:

- **mathematical analysis**, within which a closed equation system is deduced from the Petri Net and this equation system can be solved in order to acquire the desired results,
- **mathematical approximation**, which is based on numeric methods of calculation being suited for Petri Nets, which cannot be solved in the form of closed equation systems,
- **simulation**, within which the flow of tokens through the net is simulated. This simulation is carried out until the desired results can be computed according to a specified confidence bound. Therefore, the relative occurrences of tokens within the single places are acquired.

Each alternative provides its specific advantages and disadvantages regarding required computational effort, achievable accuracy etc.. In case of two or more concurrently enabled deterministic transitions, mathematical analysis is not possible and simulative or approximative methods have to be applied (Lindemann, 1998).
A further advantage of Petri Nets is the availability of comfortable mathematical methods in order to determine features of Petri Nets such as the so-called liveness or the absence of deadlocks (non-resolvable blockades) (Lindemann, 1998). The associated mathematical methods are often included in the modelling tools and therefore allow a fast verification of features like absence of deadlocks.

In order to demonstrate the application of DSPNs to model communication structures a basic DSPN is depicted in Fig. 3. A simplified arbitration scheme which handles the competition of a DMA controller and a CPU for the critical resource memory interface is modelled here. The DSPN consists of two components: a section of a CPU and a DMA-controller.
In the following, two aspects of the current state and their implications for the following state of this simple net will be explained. As can be seen in Fig. 3 the *memory request* place of the CPU and the *memory access granted* place are connected to the immediate transition ② via regular arcs. These two places are the only places which are connected to this transition and both are marked with tokens. Thus, the transition is going to fire immediately. The mentioned places are going to be cleared and the *memory access* place of the CPU is marked. This transition example describes the situation where the CPU requests the memory at a time where the *memory access* is available. The CPU accesses the memory and transfers data from or to the memory. The resource memory is therefore busy until the deterministic transition ③ fires and the place *memory access granted* is marked again. Thus, access to the memory by another device (here the DMA-controller) cannot be granted.

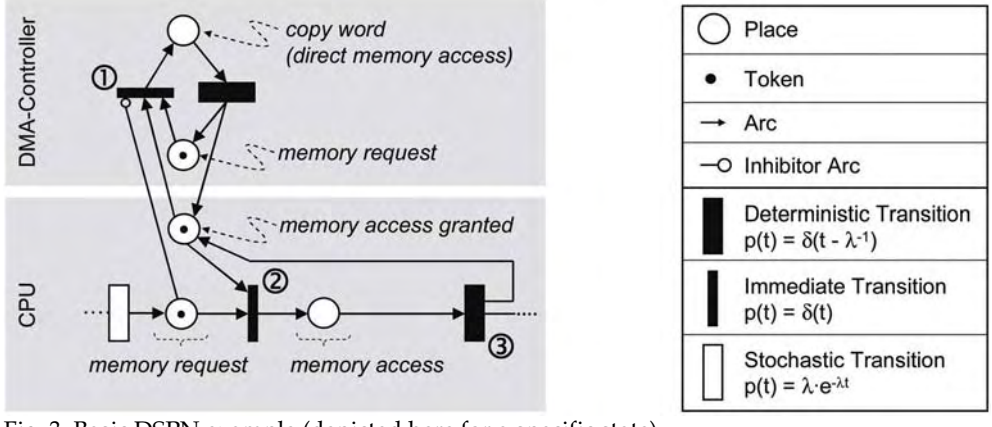


Fig. 3. Basic DSPN example (depicted here for a specific state)

The upper immediate transition ① of the DSPN depicted in Fig. 3 behaves differently compared to transition ② as one of the three places connected to transition ① is connected via an inhibitor arc. Therefore, this transition is not going to fire as long as all three connected places are marked. In case that the *memory request* place of the CPU is not marked and the other ones are marked, transition ① will fire immediately. Thus, the DMA-controller only gets access to copy a word if the CPU is not having or requesting memory access. Therefore, in this arbitration scheme the CPU has higher priority than the DMA.
The described DSPN requires input parameters such as the memory access delay time $T_③$ etc. to determine probabilities and expectations of previously defined places as mentioned above.
A variety of DSPN modelling environments is available today (Petri Nets World, 2007). For the DSPN modelling experiments described in this chapter, the modelling environment DSPNexpress (DSPNexpress, 2003) has been applied. DSPNexpress provides a graphical editor for DSPN models, as well as a solver backend for analysis of DSPNs. Experiments can

be performed for a fixed parameter set and for a parameter sweep across a user-defined range of values. The package supports the computation of the transient response e.g. the distribution of tokens (using Picards Iteration Algorithm) as well as computation of the steady state behaviour of the DSPN model. The latter can be determined by iteratively using the Generalized Minimal Residual Method, by employing the direct quadrature method or by utilizing the discrete event simulator backend (Lindemann, 1998). These methods correspond to the DSPN computation methods mentioned in the beginning of this section.

2.2 Coloured Petri nets

In this section a short introduction to Coloured Petri Nets (CPN) and to the software CPNtools (Ratzer, 2006) that has been used for the modelling examples discussed here, is given. First, the basic features of this modelling approach are presented then it is explained using a basic application example.

Coloured Petri Nets have been developed by K. Jensen in course of his PhD thesis (Jensen, 1980) to expand the modelling possibilities of classical Petri Nets. Like other forms of Petri Nets a CPN consists of places, tokens, transitions and arcs. The primary feature unique to CPNs is the inclusion of data structures into tokens. These data structures are called coloursets and resemble data structures in high level programming languages; they can range from simple data types such as integers to complex structures like structs or unions in C/C++. Similar to programming languages it is possible to define variables associated with these coloursets. Some examples of colourset and variable definitions are shown in Fig 4.a. Tokens as well as places of a CPN are always associated with a colourset and a place may only contain tokens of the same colourset as its own. Places in a CPN are depicted as ellipses (Fig 4. b) with the name of the place written into it and the associated colourset (word) below. A token in a CPN is represented by a circle (Fig 4. b). Its value (the data stored in the token) is shown in a rectangle attached to the circle. A number in the circle denotes the number of tokens with the same value. Fig 4. b for example shows a place called link associated with the colourset word and holding three tokens, two storing the value (ack, 5) one with a value of (req, 13). Tokens associated with the predefined colourset unit do not store any data and thus resemble tokens in an ordinary Petri Net or a DSPN.

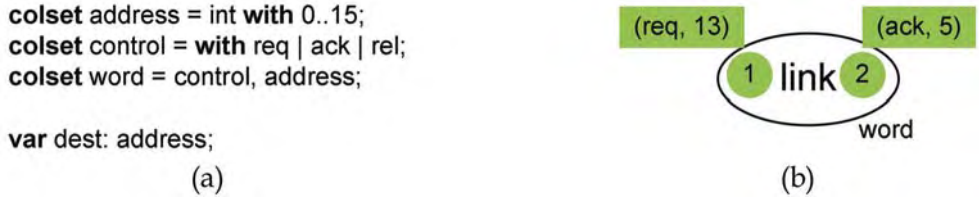


Fig. 4. Colourset and variable definitions (a) and graphical representation of a place in a CPN (b)

Transitions in a CPN are represented by rectangles (Fig. 5) and can access the data stored in tokens by mapping tokens to variables. There are two possibilities to access this data:

- **Guard conditions**: The transition is enabled only if a specific condition – called a guard condition – regarding one or more variables is met. Guard conditions are encased in brackets and written above the transition (Fig. 5a).
- **Transfer function**: The transition reads and writes variables according to a specified function that can range from simple addition of values to complex conditional commands. Transfer functions consist of the definition of *input()* variables, *output()* variables and the commands to be carried out (*action()*) and are attached below the transition (Fig. 5b).

The examples depicted in Fig. 5 show a transition that only fires if the variable ctrl has the value req (Fig. 5a) and a transition that generates an output variable dest without taking any input variables (Fig. 5b), the variable dest is filled with the return value of the function defined in the action part which in this case is a uniformly distributed random number between 0 and 15.

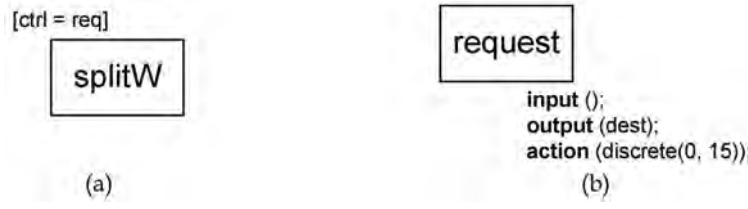


Fig. 5. Transitions with guard condition (a) and transfer function (b)

Places and transitions in a CPN are linked by arcs. Arcs in a CPN can be unidirectional like in a DSPN or bidirectional. Unidirectional arcs transfer tokens from a place to a transition or vice versa (Fig. 6 a), bidirectional arcs transfer the same token from a place to a transition and back (Fig. 6 b). Arc inscriptions define the mapping of tokens to variables. An inscription can either be a constant value (Fig. 6 a) or a variable of the colourset that is associated to the place the arc is connected to (Fig. 6 b). In case of complex coloursets an inscription can also contain a set of variables. The *word* colourset defined in Fig. 4 a for example consists of two parts, a *control* and an *address* part. A token of the colourset *word* can be either mapped to a single variable of *word* or to a set *(var1, var2)* with *var1* having the colourset *control* and *var2* being of the colourset *address*.

If all places connected to a transition by unidirectional input arcs or by bidirectional arcs hold tokens and its (optional) guard condition is met, the transition is said to be enabled. In case of more than one enabled transition in a CPN the one to fire is chosen randomly. Upon firing a transition deletes the appropriate tokens from input places and generates tokens in its output places. Places linked to the transition by bidirectional arcs are treated as both input and output places.

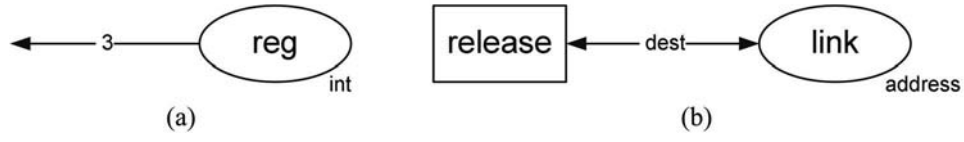


Fig. 6. Unidirectional arc with mapping to value *3* (a), bidirectional arc with mapping to variable *dest* (b)

For an analysis of clocked systems it is possible to define *timed* coloursets, defined by the keyword *timed* (Fig. 7a) and transition delays marked by the characters @+ (Fig. 7b). If a colourset is defined as *timed*, a timestamp is added to the tokens of this colourset. The timestamp cannot be accessed by guard conditions or transfer functions. When using *timed* coloursets the firing of transitions depends on a global clock counter. Transitions can only fire if the clock value is the same as the largest timestamp of its input tokens. When a *timed* transition fires, the timestamp of its output tokens is the sum of the current clock value and the transition delay, in the example in Fig. 7b this delay is 100 clock cycles.

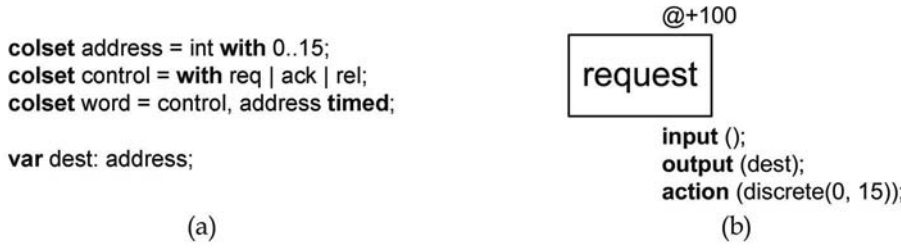


Fig. 7. Timed colourset definition (a) and transition with associated delay (b)

As an introductory example to CPN modelling a basic model of NoC communication is presented in the following paragraph. Clients in the NoC are identified by their addresses (here, integers ranging from 0 to 15). Since the communication in this NoC is based on line switching a route from source to destination has to be established before starting data transmission. The coloursets and variables used in this example are those shown in Fig. 7 a as well as the colourset unit. Messages sent through the NoC are represented by tokens of the colourset word. This colourset contains a part with the colourset control that designates how the message is to be handled and a destination address. Possible values for the control colourset are req (request route), ack (acknowledge route) and rel (release route).

In the beginning, the data source in the modelling example depicted in Fig. 8 is idle – no data is to be sent. The global clock (clock counter) is supposed to be 0. The place idle is marked, thus the transition request is enabled. This transition then fires and generates a token in the place wait – the source is now waiting for establishing of the route. At the same time the transition generates a token (req, dest) @ 100 in the place link, with @ 100 denoting the timestamp. This is a request to the network to make a route available from the source to the client with the address dest. The value of dest is a random number between 0 and 15 generated by the transfer function of the transition request (input (); … discrete(0, 15));) (see Fig. 8). With a token (req, dest) in link the transition routing becomes enabled. It fires as soon as the clock reaches 100 and generates an acknowledgement to notify the source of successful routing. Supposing the routing takes Troute = 30 clock cycles the token generated in link is (ack, empty) @ 130. Transition ack is now enabled and fires at a clock value of 130 generating a token in the place send. This means that the source switches from wait to the send mode (data transmission). Because the colourset associated with send is unit timed rather than unit like for idle and wait the token generated in send receives a time stamp of 130+Tburst, where Tburst describes the duration of a data burst. The transition release therefore cannot fire until the clock value is 130+Tburst. Transmission of a data burst is modelled only by setting the source to send mode for Tburst clock cycles. After sending the data burst (global clock at 130+Tburst) the transition release fires. Firing of this transition

resets the source state to idle and generates a token (rel, empty) in the place link, signalling the network to release the route as it is no longer needed. The rel token enables transition relNet that handles the actual release of the route, which is not modelled explicitly.

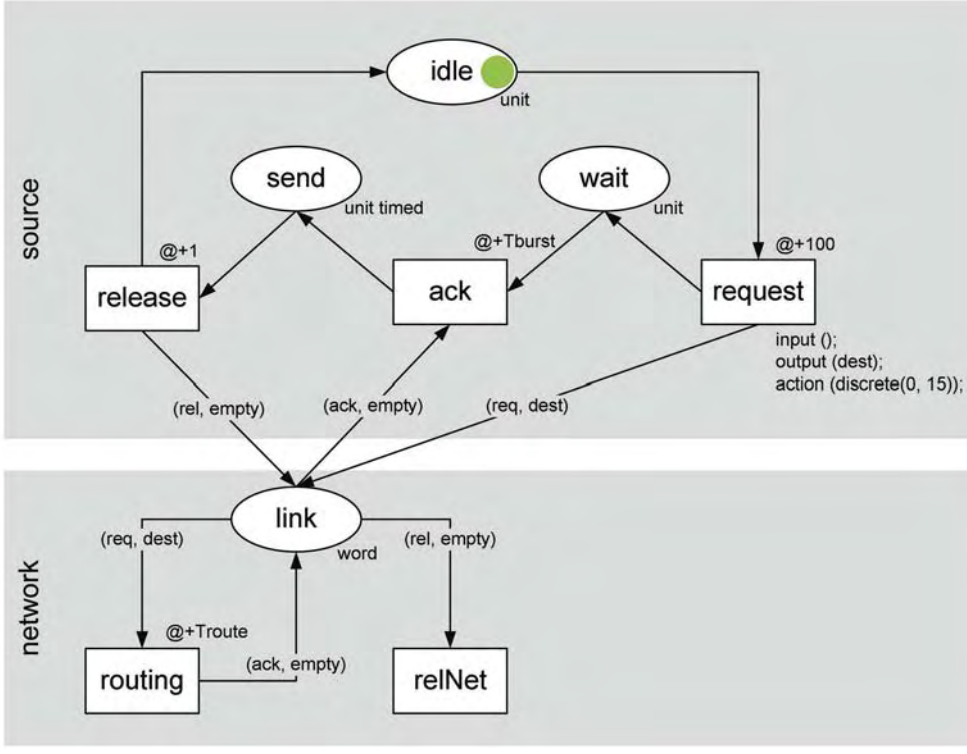


Fig. 8. CPN model of communication between a network and an attached data source

This example shows that the inclusion of data structures into CPN modelling increases the modelling capabilities compared to DSPNs. Both, the inclusion of data structures and the related use of transfer functions allow for greater functionality and smaller models that are easier to handle. With a DSPN model for example it would not be possible to store destination address information in a token or generate random addresses. In a DSPN it would be necessary to store the address in a binary format in a number of places while random generation of an address needs a sophisticated DSPN for modelling this process.

The software tool CPNtools (Ratzer, 2006), which has been used for NoC performance analysis, is a package for modelling and simulation with CPN. It consists of a graphical user interface for composition of CPN models and a simulator. CPN models are described in a format derived from Standard Markup Language (SML) called CPNml. Furthermore, CPNtools allows hierarchical definition of CPNs to facilitate reuse and simplify handling of large models. Parts of a model that are used multiple times can be encapsulated in a submodel. These submodels are included in higher hierarchy levels as substitute transitions with a defined mapping of input and output places of the transition to places in the submodel. In contrast to DSPNexpress CPNtools does not provide a means of analytical or

iterative solution but is centred on simulation. In principle it is possible to generate an ordinary Petri Net with the same functionality as a CPN that can then in turn be solved analytically. Due to the complex data structures (coloursets) and transfer functions included in a CPN the equation system describing such an underlying Petri Net would be very large. Model parameters can be measured by definition of monitors that collect data relating to different parts of the CPN such as occupation of places or the number of times a specific transition fires. The markup language used for model description also allows to use more complex monitors, including for example conditional data collection.

3. Petri net modelling of exemplary communication scenarios

In this section the exemplary application of Petri Nets for modelling communication scenarios is presented. The modelling possibilities span from simple bus based processor communication scenarios to complex NoC examples.

3.1 DSPN based processor communication model

The TMS320C6416 (Texas Instruments, 2007) (see
Fig. 9) is a high performance digital signal processor (DSP) based on a VLIW-architecture. This DSP features a couple of interfaces, an Enhanced DMA-controller (EDMA) handling data transfers and two dedicated coprocessors (Viterbi and Turbo decoder coprocessor). Exemplary communication scenarios on this DSP have been modelled. The C6416 TEB (Test Evaluation Board) platform including the C6416 DSP has been utilized to measure parameters of these modelled communication scenarios described in the following. Thus, modelling results have been proved and verified by comparison with measured values.

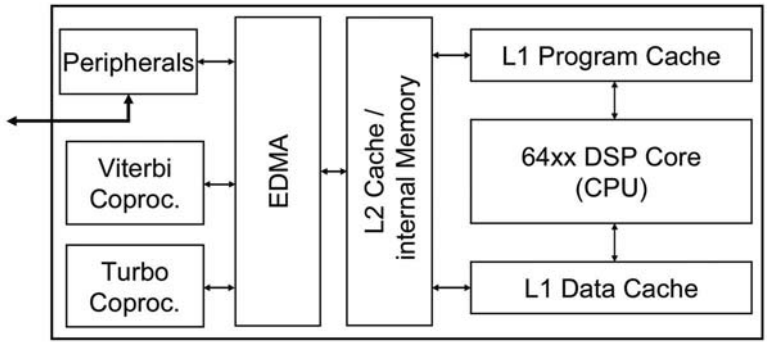


Fig. 9. Basic block diagram of the TMS320C6416 DSP

In Fig. 10 a block diagram of the C6416 and different communication paths of basic communication processes (①,② and ③) are depicted.
In the first scenario two operators compete for one critical resource, the external memory interface (EMIF). Requests for the external memory and with it the memory interface are handled and arbitrated by the enhanced direct memory access controller (EDMA) applying an arbitration scheme which is based on priority queues including four different priorities.

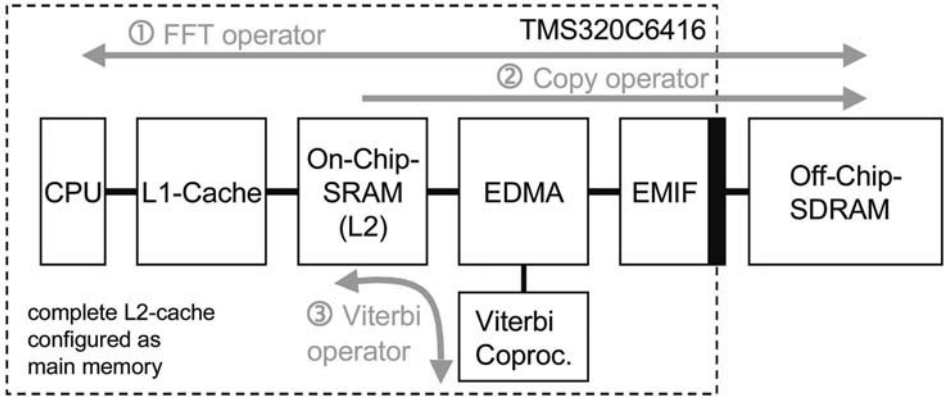


Fig. 10. Communication paths on the C6416 of different analysis scenarios

An FFT (Fast Fourier Transformation) operator runs on the CPU and reads and stores data from the external memory (e.g. for a 64-point FFT, 1107 read and 924 write operations are required which can be determined by analysis of the corresponding C-code). The corresponding communication path ① of this operator is illustrated on top of the simplified schematic of the C6416. The communication path of the copy operator ② is also depicted in Fig. 10. This operator utilizes the so called Quick Direct Memory Access mechanism (QDMA) which is a part of the EDMA. It copies data from the internal to the external memory section. Here, it requests a copy operation every CPU cycle. Since both operators run concurrently, both aim to access the critical external memory interface resource. Requests are queued in the assigned transfer request queue according to their priority. If the CPU and the QDMA both simultaneously request the memory with the same priority, the CPU request will be handled at first. In all modelled communication scenarios the priority of request initiated by the CPU and the QDMA were both assigned to the same priority which means that a competition situation for this waiting queue has been forced. The maximal number of waiting requests of this queue is 16.

The DSPN depicted in Fig. 11 represents the concurring operators and the arbitration of these two operators for the memory resource. It is separable into three subnets (see dashed boxes: Arbitration, FFT on CPU and QDMA-copy operator). The QDMA-copy operator works similar to the DMA-controller device depicted in Fig. 3.

The proprietary transfer request queue is modelled by the place *TransferRequestQueue*. The depth of the queue is modelled by inhibiting arcs with the weight 16 (the queue capacity) originating from this place. This means that these arcs inhibit the firing of transitions they are connected to if the corresponding place (*TransferRequestQueue*) is marked with 16 tokens. These inhibiting arcs are linked to subnets representing components of the system which apply for the transfer request queue. The deterministic transition T6 repetitively removes a token with a delay which corresponds to the duration of an external memory access (see parameterization in the following).

The QDMA copy operator is modelled by a subnet which produces a memory request to the EDMA every CPU cycle. The delay of deterministic transition T5 corresponds to the CPU cycle time. The places belonging to this subnet are *COPY_Start* and *COPY_Submitted*. The token of the place *COPY_Start* is removed after the deterministic delay assigned to

transition T5. The places *COPY_Submitted* and *TransferRequestQueue* are then both marked with a token. If no FFT request initiated by the CPU is pending this process recurs.

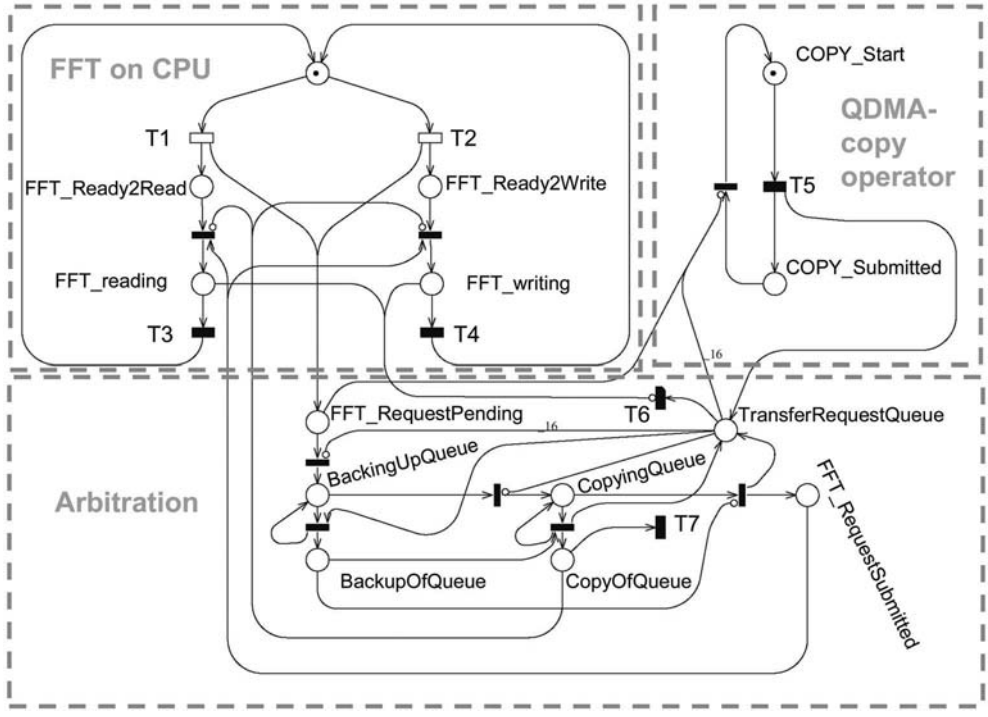


Fig. 11. DSPN of FFT / copy operator resource conflict scenario

The subnet representing the FFT operator executed on the CPU (FFT on CPU) is depicted in the upper left of Fig. 11. If one of the places *FFT_Ready2Read* (connected to stochastic transition T1) or *FFT_Ready2Write* (connected to stochastic transition T2) is marked the place *FFT_RequestPending* is also marked by a token. Hereby, a part of the model is activated which represents the queuing of the CPU requests and the assignment of the associated memory access. Places belonging to this part are: *FFT_RequestPending*, *BackingUpQueue*, *BackupOfQueue*, *CopyingQueue*, *CopyOfQueue* and *FFT_RequestSubmitted*. The place *CopyOfQueue* is a copy of the place *TransferRequestQueue*. That means that these places are marked identically. This copy proceeds by firstly removing every token in *TranferRequestQueue* and transferring it via an immediate transition to the place *BackUpQueue*. This procedure is controlled by the place *BackingUpQueue*. As soon as every token is transferred the place *CopyingQueue* is marked. Now every token in the *BackUpQueue* place is transferred simultaneously to *TransferRequestQueue* as well as to *CopyOfQueue*. Thus, the original marking of *TransferRequestQueue* is restored and also copied in the *CopyOfQueue* place. Now the *FFT_RequestSubmitted* is marked and an additional token is added to the *TransferRequestQueue* representing a further CPU request. The transitions between *FFT_RequestSubmitted* and *FFT_Reading* as well as *FFT_Writing* remove the token from the first mentioned place as soon as the CPU request is granted. The deterministic transition T7

detracts tokens from *CopyOfQueue* in the same way T6 does in context with *TransferRequestQueue*. The external memory access requested by the CPU is granted when the *CopyOfQueue* is not marked by any token. The inhibiting arcs between *CopyOfQueue* and the transitions connected to *FFT_Reading* and *FFT_Writing* ensure that only then the duration of a read and respectively a write access is modelled with the aid of deterministic transitions T3 and T4. During memory access initiated by the CPU no further request to the memory is processed. This is modelled by the inhibiting arcs originating in *FFT_Reading* and *FFT_Writing* (connected to T6). Thus, no further token from the *TransferRequestQueue* is removed.

The required parameters of the deterministic and stochastic transitions T1-T7 of this DSPN model are given in Table 1.

Here, it holds:

T_{FFT}: duration of a single block FFT operation (dependent on FFT length, without parallel copy operation)

$N_{Read/Write}$: number of memory read/write accesses per FFT operation

$T_{Read/Write,ext.mem}$: time required to read/write a word from/to the external memory

$p_i(t)$: probability density function of the delay time of a specific transition

Transition	Transition type	Formula and parameters
T1	stochastic (negative exponential distributed)	$p_1(t) = \lambda_1 \cdot e^{-\lambda_1 t}$ for $t > 0$ with $\lambda_1 = \frac{N_{Read}}{T_{FFT} - N_{Read} \cdot T_{Read,\,ext.mem} - N_{Write} \cdot T_{Write,\,ext.mem}}$
T2	stochastic (negative exponential distributed)	$p_2(t) = \lambda_2 \cdot e^{-\lambda_2 t}$ for $t > 0$ with $\lambda_2 = \frac{N_{Write}}{T_{FFT} - N_{Read} \cdot T_{Read,\,ext.mem} - N_{Write} \cdot T_{Write,\,ext.mem}}$
T3	deterministic	$\Delta t_3 = T_{Read,ext.\,mem} = N_{Read} \cdot 0.188\,\mu s$
T4	deterministic	$\Delta t_4 = T_{Write,ext.\,mem} = N_{Write} \cdot 0.088\,\mu s$
T5	deterministic	$\Delta t_5 = 1/f_{Proc} = 1/500\,MHz = 2\,ns$
T6	deterministic	$\Delta t_6 = 1/f_{ext.mem} = 1/133\,MHz = 7.5\,ns$
T7	deterministic	$\Delta t_7 = 1/f_{ext.mem} = 1/133\,MHz = 7.5\,ns$

Table 1. Transition type and transition parameters of the DSPN model of Fig. 11

The required input parameters for the DSPN model like the duration of a single block FFT without running the concurrent copy operator (T_{FFT}) have been determined by

measurements performed on a DSP board. In order to verify the assumptions e.g. for $T_{Read,ext.mem}$ and $T_{Write,ext.mem}$, several experiments with a variation of external factors have been performed. For example, the influence of the refresh frequency has been studied. By modification of the value within the so-called EMIF-SDTIM register the refresh frequency of the external SDRAM could be set. Through different measurements it could be verified that the resulting influence on the read and write times is below 0.3 % and therefore negligible. For the final measurements a refresh frequency of 86.6 kHz (what is equal to a refresh period of 1536 memory cycles and therefore an EMIF-SDTIM register value of 1536) has been applied.

The influence of the parameter N_{Read} will be explained exemplarily in the following. The probability density function $p_1(t)$ which is a function of N_{Read} characterizes the probability for each possible delay of the stochastic transition T1. N_{Read} directly influences the expected delay respectively the firing probability of T1. Here, high values for N_{Read} correspond to a low firing probability respectively a large expected delay and vice versa.

The modelling results of the DSPN for the duration of the FFT are depicted in Fig. 12. Here, the calculation time of the FFT operator determined by simulation with the DSPN model has been plotted against different FFT lengths. In order to attain a quantitative evaluation of the computed FFT's duration, reference measurements have been made again on a DSP board. As can be seen from Fig. 12 the model yields a good estimation of the duration for the FFT operator. The maximum error is less than 10 % (occurring in case of an FFT length of 1024 points).

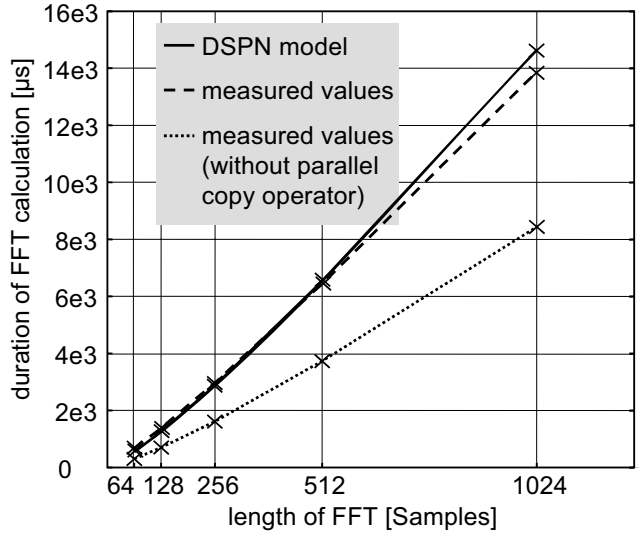


Fig. 12. Comparison of measured values with DSPN (FFT vs. copy operator)

Another example based on this DSP was analyzed in order to consolidate the suitability of using DSPNs for modelling in terms of on-chip communication: Now, the Viterbi Coprocessor (VCP) and the copy operator compete for the critical external memory interface resource. The VCP also communicates with the internal memory via the EDMA (commu-

nication path ③ in Fig. 10). Arbitration is handled by a queuing mechanism configured here in that way that only a single queue is utilized. This is accomplished by assigning the same priority to all EDMA requestors i.e. memory access is granted to the VCP and the copy operator according to a first-come-first-serve policy.
For this experiment the VCP has been configured in the following way. The constraint length of the Viterbi decoder is 5, the number of states is 16 and the rate is 1/2. In the VCP configuration inspected here, the VCP communicates with the memory by getting 16 data packages of 32x32 bit in order to perform the decoding. Both, EDMA and VCP are clocked with a quarter of the CPU clock frequency (fCPU = 500 MHz). The results are transferred back to the memory with a package size of 32x32 bit. Performing two parallel operations (Viterbi decoding and copy operation), the two operators have to wait for their corresponding memory transfers. The EDMA mechanism of the C6416 always completes one memory block transfer before starting a new one. Hence, there is a dependency of the Viterbi decoding duration on the EDMA frame length. This situation has been modelled and the results have been compared to the measured values as depicted in Fig. 13.

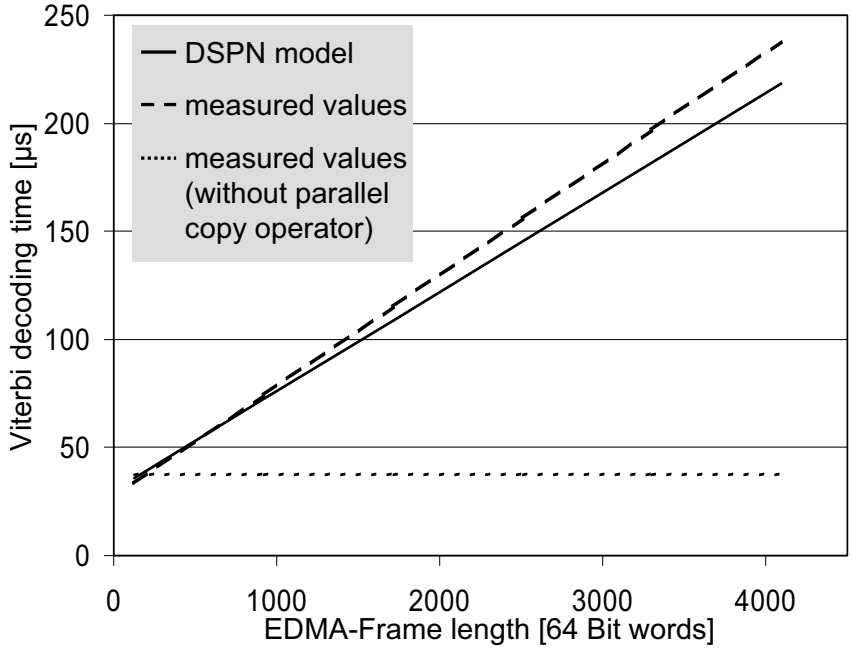


Fig. 13. Comparison of measured values with DSPN (Viterbi vs. copy operator)

Performing only the Viterbi decoding, there is of course no dependency on the EDMA frame length. If a copy operation is carried out, the Viterbi decoding time significantly increases. In detail not the decoding process itself is concerned but the duration of data package transfers between VCP and internal memory. Again the maximum error is less than 10 %.

3.2 DSPN based switch fabric communication model

The second DSPN modelling example deals with communication via a switch fabric based structure. The modelled scenario is a resource sharing conflict. This scenario has been evaluated on an APEX based FPGA development board (Altera, 2007).

A multi processor network has been implemented on this development board by instantiating Nios soft core processors on the corresponding FPGA. The synthesizable Nios embedded processor is a general-purpose load/store RISC CPU that can be combined with a number of peripherals, custom instructions, and hardware acceleration units to create custom system-on-a-programmable-chip solutions. The processor can be configured to provide either 16 or 32 bit wide registers and data paths to match given application requirements. Both data width versions use 16 bit wide instruction words. Version 3.2 of the Nios core typically features about 1100 logic elements (LEs) in 16 bit mode and up to 1700 LEs in 32 bit mode including hardware accelerators like hardware multipliers.

More detailed descriptions can be found in (Altera, 2001). A processor network consisting of a general communication structure that interfaces various peripherals and devices to various Nios cores can be constructed. The Avalon (Avalon, 2007) communication structure is used to connect devices to the Nios cores. Avalon is a dynamic sizing communication structure based on a switch fabric that allows devices with different data widths to be connected with a minimal amount of interfacing logic. The corresponding interfaces of the Avalon communication structure are based on a proprietary specification provided by Altera (Avalon, 2007). In order to realize a processor network on this platform the so-called SOPC (system on a programmable chip) Builder (SOPC, 2007) has been applied. SOPC is a tool for composing heterogeneous architectures including the communication structure out of library components such as CPUs, memory interfaces, peripherals and user-defined blocks of logic. The SOPC Builder generates a single system module that instantiates a list of user-specified components and interfaces incl. an automatically generated interconnect logic. It allows to modify the design components, to add custom instructions and peripherals to the Nios embedded processor and to configure the connection network.

The analyzed system is composed of two Nios soft cores which compete for access to an external shared memory (SRAM) interface. Each core is also connected to a private memory region containing the program code and to a serial interface which is used to ensure communication with the host PC. The proprietary communication structure used to interconnect all components of a Nios based system is called Avalon which is based on a flexible crossbar architecture. The block diagram of this resource sharing experiment is depicted in Fig. 14. Whenever multiple masters can access a slave resource, SOPC Builder automatically inserts the required arbitration logic. In each cycle when contention for a particular slave occurs, access is granted to one of the competing masters according to a Round Robin arbitration scheme. For each slave, a share is assigned to all competing masters. This share represents the fraction of contention cycles in which access is granted to this corresponding master. Masters incur no arbitration delay for uncontested or acquired cycles. Any masters that were denied access to the slave automatically retry during the next cycle, possibly leading to subsequent contention cycles.

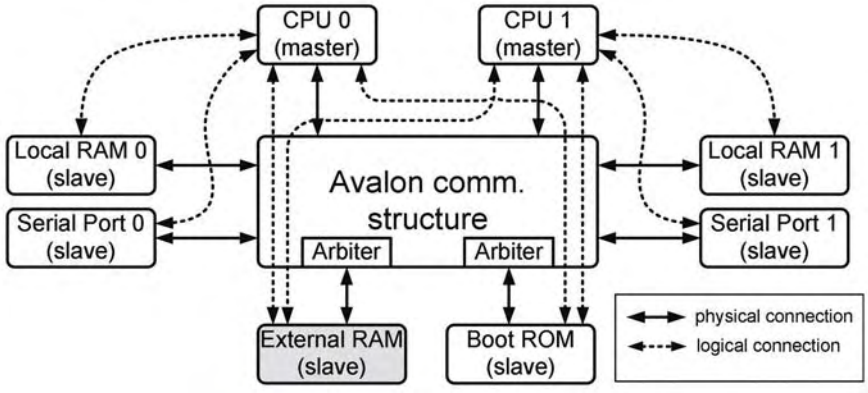


Fig. 14. Block diagram of the resource sharing experiment using the Avalon communication structure

In the modelled scenario the common slave resource for which contention occurs is a shared external memory unit (shaded in gray in Fig. 14) containing data to be processed by the CPUs. Within the scope of this fundamental resource sharing scenario several experiments with different parameter setups have been performed to prove the validity of the DSPN modelling approach. Adjustable parameters include:

- the priority shares assigned to each processor,
- the ratio of write and read accesses,
- the mean delay between memory accesses.

These parameters have been used to model typical communication requirements of basic operators like digital filters or block read and write operations running on these processor cores. In addition, an experiment simulating a more generic, stochastic load pattern, with exponentially distributed times between two attempts of a processor to access the memory has been performed. Here, each memory access is randomly chosen to be either a read or a write operation according to user-defined probabilities. The distinction between load and store operations is important here because the memory interface can only sustain one write access every two cycles. Whereas no such limitation exists for read accesses. The various load profiles were implemented in C, compiled on the host PC and the resulting object code has been transferred to the Nios cores via the serial interface for execution. In the case of the generic load scenario, the random values for the stochastic load patterns were generated in a MATLAB routine. The determined parameters have been used to generate C code sequences corresponding to this load profile. The time between two attempts of a processor to access the memory has been realized by inserting explicit NOPs (No Operation instruction) into the code via inline assembly instructions. Performance measurements for all scenarios have been achieved by using a custom cycle-counter instruction added to the instruction set of the Nios cores. The insertion of NOPs does not lead to an accuracy loss related to pipeline stalls, cache effects or other unintended effects. The discussed example is constructed in such a way that these effects do not occur. In a first step, a basic DSPN model has been implemented (see Fig. 15) in less than two hours. Implementation times of the DSPN models are related to the effort a trained student (non-expert) has to spend to realize the corresponding model. The training time for a student to become acquainted with DSPN modelling lasts a couple of days. Distinction between read and write accesses was explicitly

neglected to achieve a minimum modelling complexity. The DSPN consists of four sub-structures:

- two parts represent the load generated by the Nios cores (CPU #1 and #2)
- a basic cycle process subnet providing a clock signal (Clock-Generation)
- the more complex arbitration subnet

Altogether, this basic model includes 19 places and 20 transitions. The immediate transitions T1, T2 and T3 and the associated places P1, P2 and P3 (see Fig. 15) are an essential part of the Round Robin arbitration mechanism implemented in this DSPN. The marked place P2 denotes that the memory is ready and memory access is possible. P1 and P3 belong to the CPU load processes and indicate that the corresponding CPU (#1, #2) tries to access the memory. If P1 and P2 or P3 and P2 are tagged the transition T1 or accordingly transition T3 will fire and remove the tokens from the connected places (P1, P2 or P2, P3). CPU #1 or CPU #2 has been assigned the memory access in this cycle. A collision occurs if P1, P2 and P3 are tagged with a token. Both CPUs try to access the memory in the same cycle (P1 and P3 marked). Furthermore, the memory is ready to be accessed (P2 marked). A higher priority has been assigned to transition T2 during the design process. This means that if the conditions for all places are equal the transition with the highest priority will fire first. Therefore, T2 will fire and remove the tokens from the places. Thus, the transitions T1, T2 and T3 and the places P1, P2 and P3 handle the occurrence of a collision.

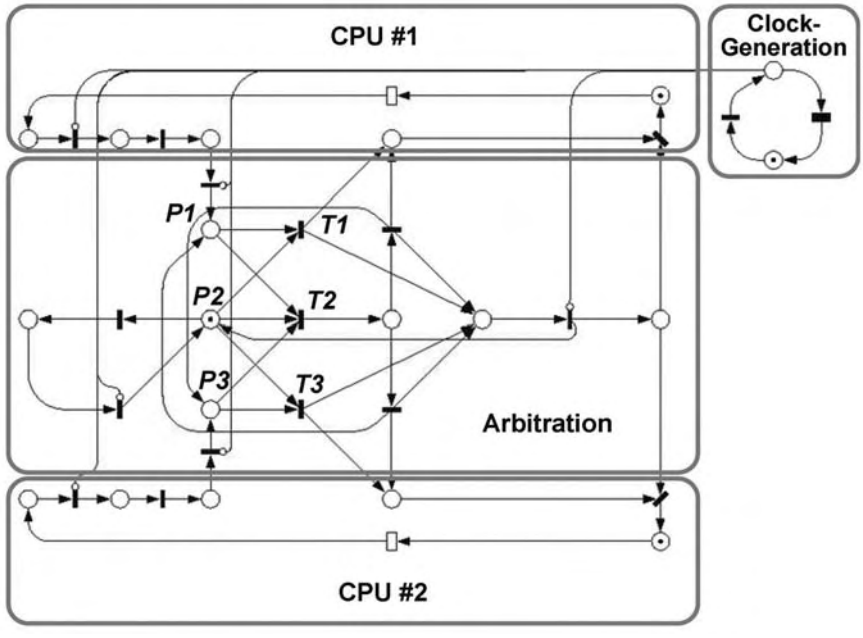


Fig. 15. Basic DSPN for Avalon-Nios example

The modelling results discussed in the following have been acquired by application of the iterative evaluation method. Though the modelling results applying this basic DSPN model are quite accurate (relative error less than 10 % compared to the physically measured values, see Fig. 18), it is possible to increase the accuracy even more by extending the modelling

effort for the arbitration subnet. For example it is possible to design a DSPN model of the arbitration subnet which properly reflects the differences between read and write cycles. Thus, the arbitration of write and read accesses has been modelled in different processes resulting in different DSPN subnets. This results in a second and enhanced DSPN model depicted in Fig. 16. The implementation of this enhanced model has taken about three times the effort in terms of implementation time (approximately five hours) than the basic model described before.

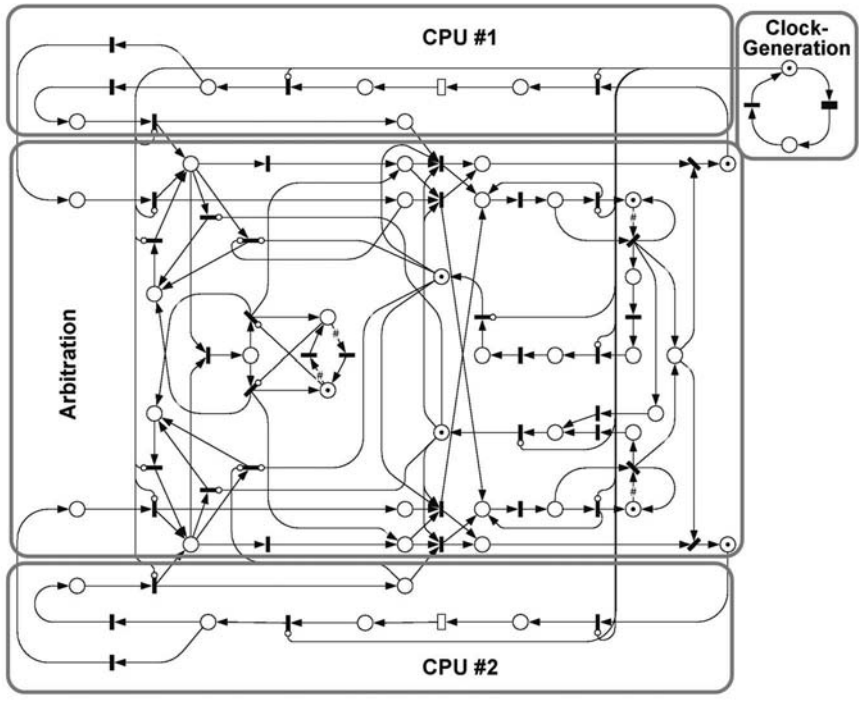


Fig. 16. Enhanced DSPN for Avalon-Nios example

The DSPN model now consists of 48 transitions and 45 places. Compared to the basic model the maximum error has been further reduced (see Fig. 17 and Fig. 18). The enhanced model also properly captures border cases caused e. g. by block read and write operations.
The throughput measured for a code sequence containing 200 memory access instructions has been compared to the results of the basic and enhanced DSPN model. Fig. 18 shows the relative error for the throughput (results of the DSPN model compared to measured results of an FPGA based testbed) which is achieved by varying the mean number of computation cycles between two attempts of a processor to access the memory. On average the relative error of calculated memory throughput is reduced by 4-6 % with the transition from the basic to the enhanced model. Using the enhanced DSPN model the maximum estimation error is below 6 %. As mentioned before, the evaluation of DSPNs can be performed by different methods (see Fig. 19). The effort in terms of computation time has been compared for a couple of experiments. Generally, the time consumed when applying the simulation

method is about two orders of magnitude longer than the time consumed by the analysis methods. The simulation parameters have been chosen in such a way that the simulation results match the results of the analytic approaches. DSPNexpress provides an iterative method (Picard's iteration method) and a direct solution method (general minimal residual method). Fig. 19 illustrates a comparison of the required computational time for the analysis and the simulation of the introduced basic and enhanced DSPN models. For the example of the enhanced model the computation time of the DSPN analysis method only amounts to 0.3 sec. and the DSPN simulation time (10^7 memory accesses) amounts to 20 sec. on a Linux based PC (2.4 GHz, 1 GByte of RAM). The difference between the iterative and direct analysis method is hardly noticeable.

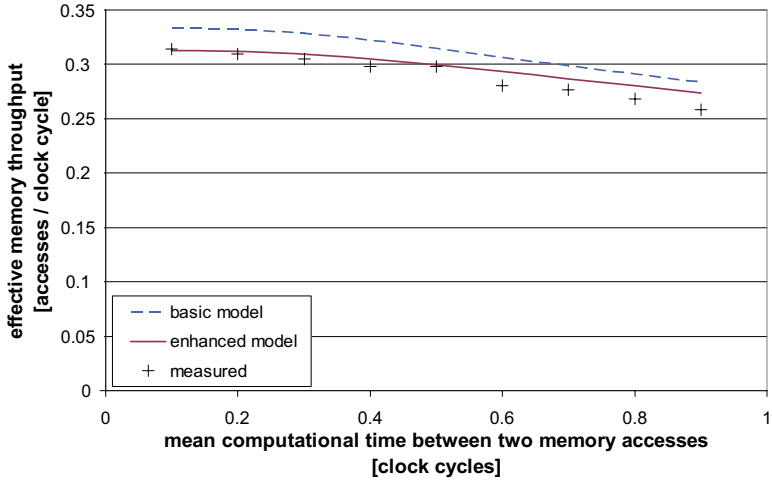


Fig. 17. Effective memory throughput comparison

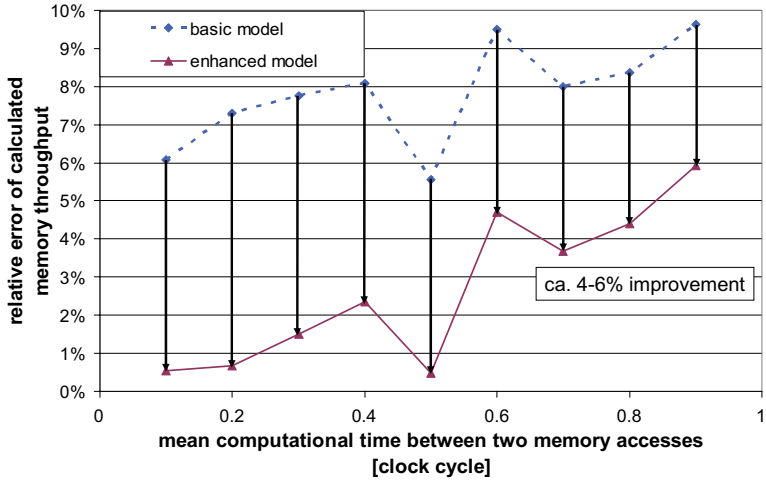


Fig. 18. Relative error of memory throughput for basic and enhanced DSPN model

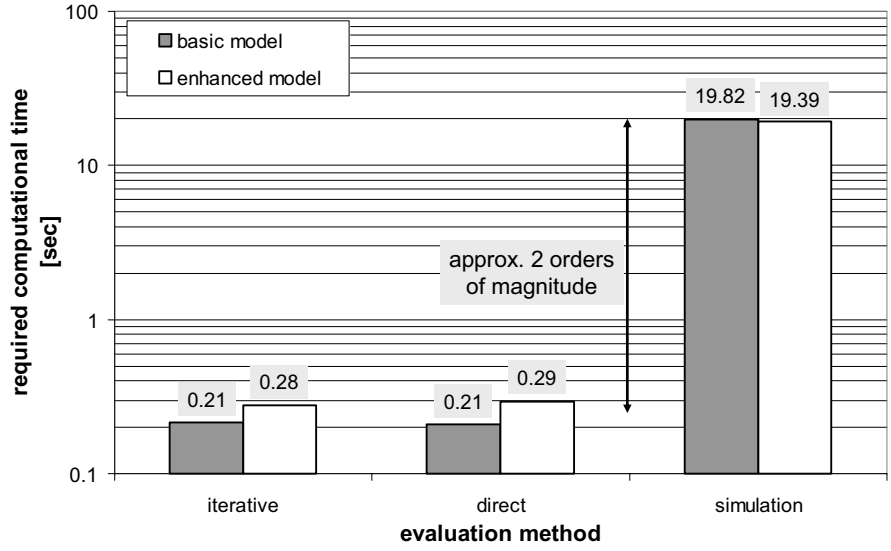


Fig. 19. Required computational effort for the different evaluation methods

3.3 CPN based NoC model

The NoC model presented in this section consists of 25 network nodes arranged in a 5x5 square mesh as depicted in Fig 20. Each network node consists of a routing switch and a client. Clients are any data sources connected to the NoC, for example embedded processors. They are identified by a unique address containing their x and y coordinates. The routing switches and the links connecting them form the actual communication infrastructure facilitating communication between clients. The switching scheme chosen in the model is line switching. Hence, communication between any two clients can be divided into three stages:

- establishing of a route from originating client (source) to the receiving one (destination),
- data transmission and
- releasing the route.

The communication protocol is described in more detail below. The implemented routing algorithm is xy-routing. This is a minimal-path routing algorithm trying first to route horizontally until the destination column is reached (matching of x-coordinates) and then completes the route vertically (matching of y-coordinates). The arbitration scheme used is first-come-first-serve.

For analyzing different traffic experiments the clients are configurable by

- a list of possible destinations for communication attempts,
- the duration of data bursts (measured in clock cycles) to be transmitted (*Lburst*) and
- the average delay between the end of a transmission and the request for routing the next one (*Ldelay*).

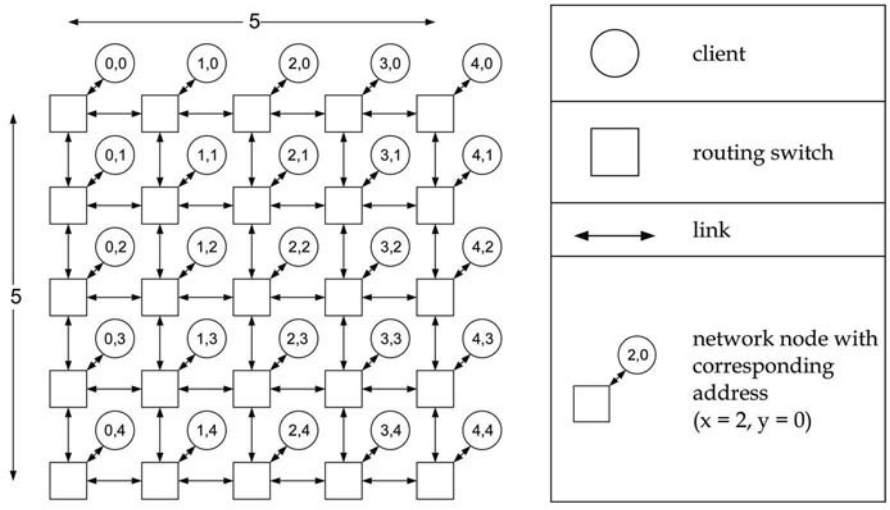


Fig. 20. NoC setup for experiments

The routing switches are not configurable. Performance is measured by latency and source load. Latency in this case corresponds with the time needed for establishing a route and is chosen as a performance measure because it is critical for applications that need fast data transmission, for example real time applications. For applications generating a lot of data, throughput is important. Therefore, the source load is selected as another performance characteristic. The source load is defined as the relative time a data source is transmitting; the requested source load is defined as

$$source\_load_{req} = \frac{Lburst}{Lburst + Ldelay} \quad , \tag{1}$$

while the achieved source load takes into account the latency (*Lat*) caused by establishing a route:

$$source\_load_{ach} = \frac{Lburst}{Lburst + Ldelay + Lat} \, . \tag{2}$$

Since the requested load does not include latency which always occurs in a network, the achieved source load is always smaller than the requested one.

The essential parts of the model are briefly explained in the following. The NoC model consists of two main submodels, the routing switch model and the client model. Each network node consists of a routing switch and an attached client. Messages sent through the net are represented by tokens of the colourset *word* (Fig. 21). Each message consists of a header (colourset *control*) defining how it is to be processed and the *content* that is used according to the header specification. Possible headers are *req* (request route), *rel* (release route), *relb* (acknowledge release route), *kill* (routing failure) and *ack* (acknowledge route), *content* can be either a destination address (*de*) or *emp*ty.

```
colset coordinate = int with 0..15;                 // x and y coordinates of nodes
colset address = coordinate, coordinate;            // node addresses (x, y)
colset control = with req | rel | relb | kill | ack; // control sequence to identify message type
colset content = de:address | empty;                // contains either an address type variable de or is empty
colset word = control, content;                     // messages to be sent through the network
```

Fig. 21. Essential coloursets for the CPN based NoC model

Since in this example line switching is used, communication between two clients is made up of several stages as stated above. When a data source tries to send data, a *req* message is generated that is then routed through the network according to the routing algorithm. The content of a *req* message contains the destination address of the route. In each network node the *req* message is processed by the routing switch which reserves the appropriate connection of two of its ports for the requested route. This is done by comparing the local and destination addresses and then adding a member to the list of current routes stored in a token of the colourset *routelist*. Upon arrival of the *req* signal at the destination the client generates an *ack* message to travel back along the route. Reception of an *ack* at the source triggers data transmission. Data is not represented by any tokens because network performance does not depend on the actual data sent across the NoC but on the time the route is occupied. After completing data transmission the source client sends a *rel* signal which is returned by the destination as *relb*. Processing of a *relb* message initiates release of the partial routes stored in the routing switches. When routing fails because an output port in any node cannot be reserved since it is already occupied by another route, a *kill* signal is generated and sent backwards along the route. If employing a static routing algorithm this signal is handled like *relb*. If an adaptive algorithm is used processing of a *kill* message can lead to the attempt to select another route. When the source client receives a *kill* signal it will issue a new routing request (*req*) after a configurable time.

The client model is comprised of two submodels, *source* and *sink*, of which only the *source* model will be discussed here in detail as the functionality of the *sink* model is elementary. As the data content is not important for network performance this basic model is sufficient to model a wide range of possible clients that can be attached to a NoC.

The *source* submodel shown in Fig 22. includes transitions for handling incoming and outgoing messages. The place *out* is the interface of the data source to the network, similar to the place *link* in the introductory example (

Fig. 8). The current state of the source is stored in the *status* place. The source sequentially passes through the states *idle*, *wait* and *send*. These states are defined as:

- *idle*: There is currently no data to be sent.
- *wait*: A route was requested, the source is waiting for it to be established.
- *send*: A route was established, the source is transmitting data.

Switching from *idle* to *wait* occurs when the transition *request* fires. This transition generates tokens in the places *wait_for_route* and *out*. The token in *wait_for_route* is later used to measure latency, the token *(req, addr)* in the place *out* signals the network, that a route to the client with address *addr* is requested. The timestamp of the token generated in out is the current global clock value increased by a random number between one and *Lpause* due to the processing delay associated with the *request* transition. The network then replies by either signalling successful routing (*ack*) or an aborted routing attempt (*kill*) by generating a token in the place *out*. If a *kill* token is generated the transition *killreq* becomes enabled. The timestamp is increased by *Tkill* to ensure that there is a delay before the new attempt. If an *ack* token is in the place *out*, the transition *ackdata* fires. The source state is thereby set to *send*.

Firing of *ackdata* removes the token from *wait_for_route* – the route was successfully established – and generates a token into the place *sending*. This token receives a time stamp according to the configured burst length (*Tburst*). The source stays in the *send* state for *Lburst* clock cycles before the transition *release* fires and sets the status back to *idle*. The token *(rel, emp)* generated in the place *out* by this transition signals the network to release the route originating from this source. Successful release of the route is then acknowledged by generation of a *relb* token in the place out. This token enables the transition *release_back* by which it is then removed.

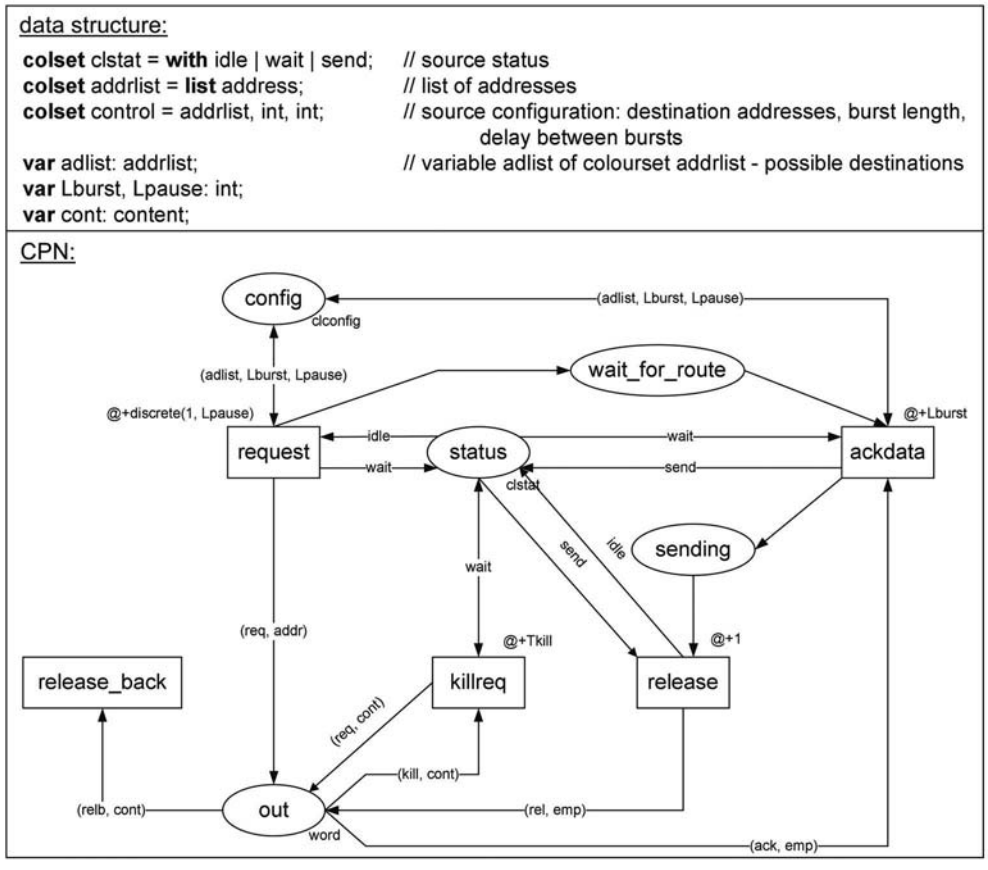


Fig. 22. CPN submodel of a data source and required data structure

The place config is used to configure the source. Variables that can be configured are:

- *adlist*: A list of destination addresses that the source can request routes to. If an address is contained in this list multiple times the probability that a route to the corresponding client is requested increases accordingly.
- *Lburst*: The length of a data burst, measured in clock cycles.
- *Lpause*: The maximum delay between the end of a transmission and the subsequent routing request.

Performance measures obtained in this submodel are the number of routing requests sent, source load and latency. The number of routing requests sent is measured by counting the times the transition *request* fires. Source load is measured by the average occupation of the place *sending*. Latency in this case corresponds to the time used for route establishment. It is measured by computing the total time spent for route establishment (product of the inspected time period and the average occupation of the place *wait_for_route*). This value is then divided by the number of routing requests sent.
The *sink* submodel is comprised of transitions to respond to incoming *req* and *rel* messages with the appropriate signals of its own, these being *ack* and *relb*, without gaining any performance measures.
The routing switch model (Fig. 23) consists of four submodels, *req*, *forward*, *ack* and *kill* as well as the places *addr*, *routes* and the interface places *inWest* to *inClient* and *outWest* to *outClient*. The *addr* place contains the address of the node in the network while the current switching table containing the connections of input and output ports is stored in *routes*. The places *inWest* to *inSouth* and *outWest* to *outSouth* are interfaces to neighbouring routing switches, the places *inClient* and *outClient* are interfaces to the attached client (*inClient* is mapped to the place *out* of the *source* model shown in Fig. 22). The *req* submodel itself contains two further submodels, *router* and *arbiter*, and is used to process routing requests. The *forward* and *ack* submodels transmit tokens according to the switching table the former processing *rel* the latter *ack* messages. The *kill* submodel resembles the *ack* model but also includes deletion of routes from the switching table for handling *relb* and *kill* signals. The *router* and *arbiter* submodels contained in the *req* model are used to separate the actual routing from the arbitration.

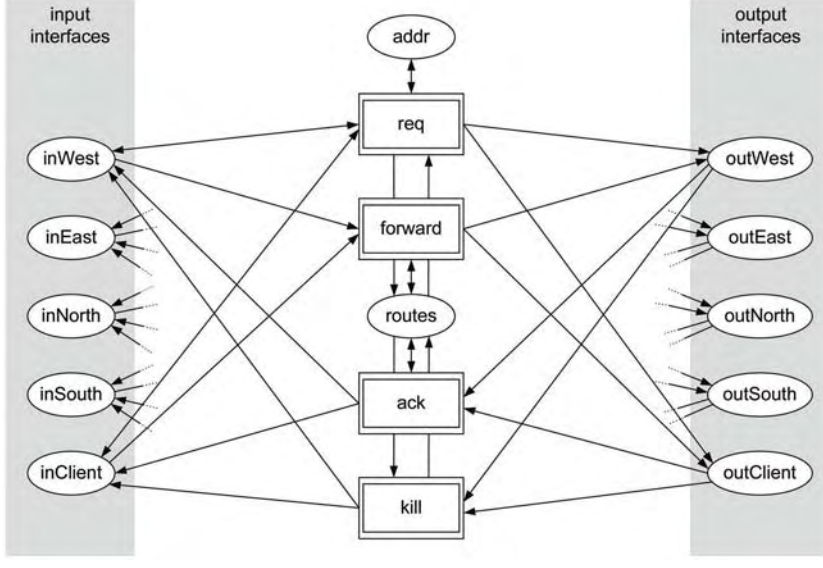


Fig. 23. CPN model of a routing switch for mesh networks

The places *inWest* and *interest* in the router model depicted in Fig. 24 are interfaces, *inW* is mapped to the equivalent place in the routing switch model while *interW* is the connection to the *arbiter* model. The section of the router shown in Fig. 24 is the part responsible for the

western port of the routing switch, the sections for the other ports resemble this section and only differ in the names of places and transitions. The place *addr* contains the address of the network node which the routing switch belongs to. In the *router* the destination address contained in an incoming *req* token (*x1*, *y1*) is compared to the address of the current network node (*x0*, *y0*) before generating a request to the *arbiter* according to the routing algorithm. In case of the router shown in Fig. 24 this is a static xy-routing scheme. The request generated by the *router* is used by the *arbiter* to make the appropriate entry in the switching table (*routes*), resource conflicts are resolved in a first come first serve manner. If an output port is occupied, a kill signal is generated by the *arbiter*. The single performance characteristic gained in the routing switch is its load, which is monitored by occupation of the place *routes*.

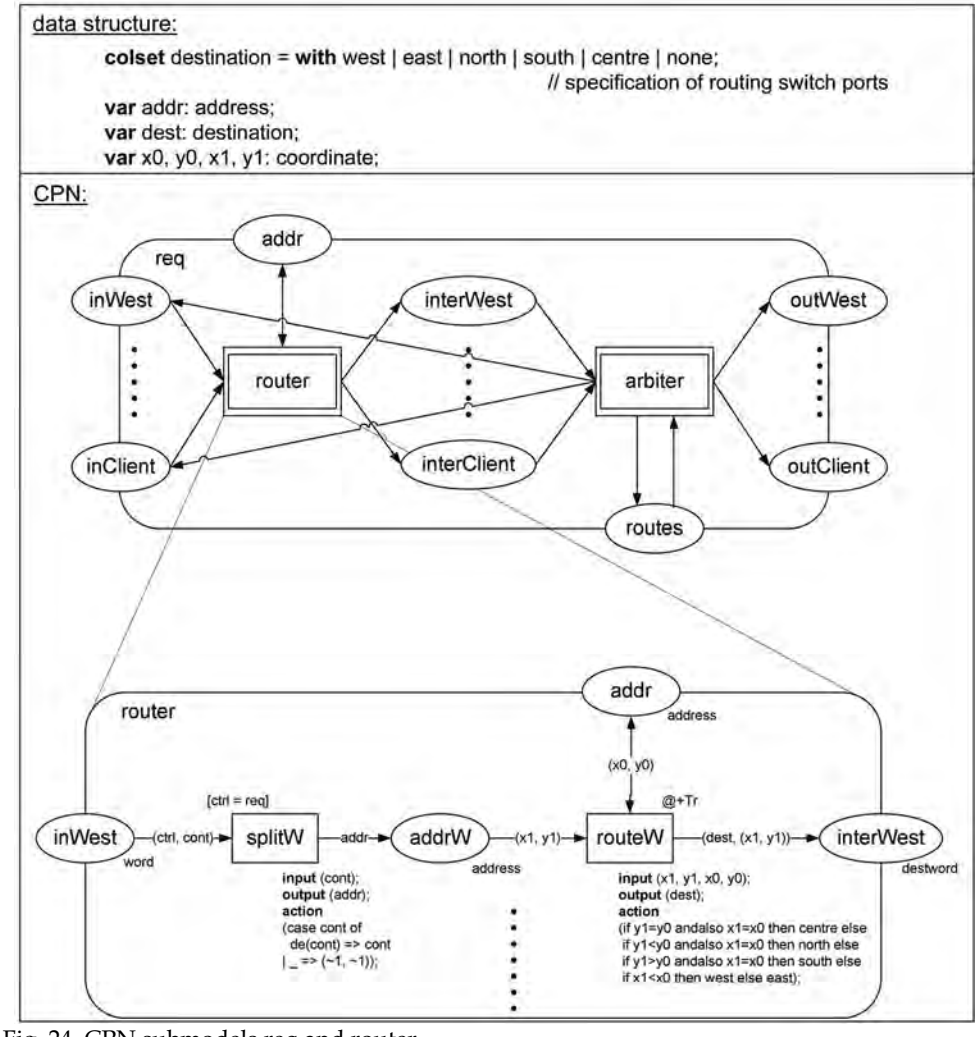


Fig. 24. CPN submodels req and router

In the context of network performance analysis three different experiments have been conducted with the model. The results have afterwards been compared to those obtained with an FPGA based NoC emulator (Neuenhahn, 2006). The experiments are defined as follows:

- **Experiment 1**: Each client is configured to send data to all other clients with equal probability. Requested source load is the same at all clients and set to values from 10 % to 80 % with steps of 10 %.
- **Experiment 2**: Same as experiment 1, but static xy-routing is exchanged for an adaptive variant. This variant tries to route vertically, if a resource conflict prohibits horizontal routing. Source load is set to the same values as in Experiment 1.
- **Experiment 3**: All clients are configured to send data to the nodes (1, 0), (3, 0) and (2, 2) with a probability of 14 %. Here, a testcase is modelled, where some I/O interfaces are accessed more often than other clients. Probability to send to another node is 3 %. The requested source load is 50 % at all clients.

The size of the complete model is 725 places and 1050 transitions for experiments 1 and 3 and 800 places and 1075 transitions in case of experiment 2 because the adaptive router is more complex than the static one. Due to the complexity contained in colousets and transfer functions these numbers are not easily comparable to those of DSPN models.
Traffic is generated in form of data burst with a duration of *Lburst = 100* clock cycles, source load is set by adjusting the delay between transmissions (*Lpause*).
Source load in the first two experiments is set to values from 10 % to 80 % with steps of 10 %. All simulations are stopped and repeated after a total of 10,000 transmissions; all results presented below are averaged over 50 repetitions. Model components are the same for all experiments with exception of the adaptive router for the second one.
All simulations were conducted on a computer with a dual Intel Pentium processor running at 3.0 GHz, 1 GB RAM and Microsoft Windows XP as operating system.
Fig. 25 shows a comparison between the first two experimental setups (experiment 1, experiment 2) with static and adaptive versions of the xy-routing algorithm. It is obvious that the achieved source load is close to the requested one for small loads as there are only few resource conflicts. With higher load and thus shorter pauses between individual transmissions the number of conflicts increases. This leads to a decline in network performance. The use of adaptive xy-routing slightly increases the achieved load because

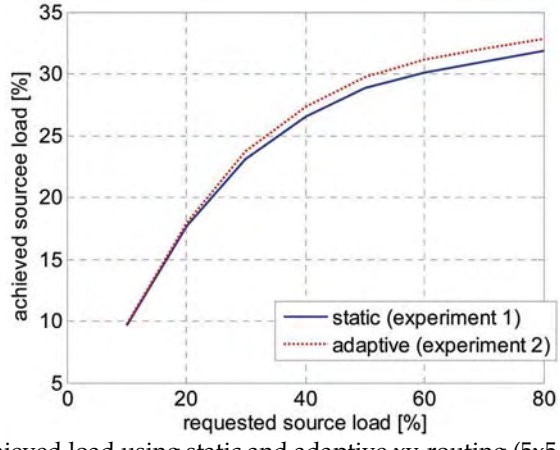


Fig. 25. Average achieved load using static and adaptive xy-routing (5x5 mesh)

some resource conflicts can be resolved without releasing a partial route and reattempting from its source. An adaptive variant that would also include rerouting attempts if a network node receives a *kill* signal would further increase performance at the cost of more complex routing switches.

By analyzing the load of the routing switches it is furthermore possible to locate hotspots that are generated by certain traffic patterns. An example of such a pattern is the one used in experiment 3. The results obtained from simulating experiment 1 and experiment 3 are shown in Fig. 26. In experiment 1 the load distribution among the routing switch shows a flat profile with its maximum in the middle of the NoC (address 2, 2). This hotspot is caused by the fact, that the majority of possible routes contains the central node because the NoC is symmetric to its centre. As all possible pairs of source and destination of routes are equally probable the load of the routing switches is determined only by the number of possible routes including the corresponding node. The modification of the traffic pattern in experiment 3 results in the forming of three distinct hotspots easily identified as peaks in the shown load diagram. Besides high switch loads at those network nodes that are accessed with a higher probability (addresses 2, 2; 1, 0; 3, 0) hotspots also form at the addresses (1, 1) and (3, 1) because most routes ending in (1, 0) and (3, 0) run through these nodes.

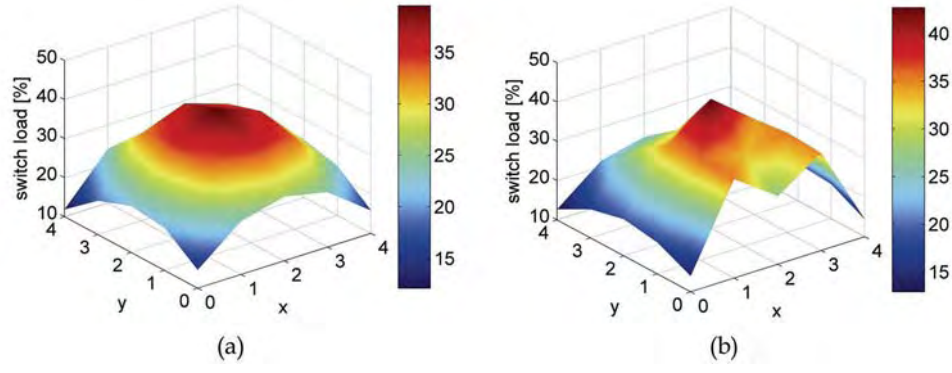


Fig. 26. Switch load for uniform (a, experiment 1) and irregular traffic distribution leading to hotspots (b, experiment 3)

A comparison of these results with an FPGA based emulator (Neuenhahn, 2006) shows that the error of the results obtained with CPN modelling is approximately 2 % for both performance measures. As an example, Fig. 27 shows the relative deviation of the CPN modelling results from those of the emulation for experiment 1. The results shown in Fig. 27 are obtained with a requested load of 20 %. For any single load setting the error is below 5 % within less than five minutes of simulation. This corresponds to approximately eight simulation runs that are needed to gain acceptably accurate result. Complete syntax checking takes approximately 15 minutes and is only needed before the first simulation run. Since syntax checking in CPNtools is incremental this time is reduced to one minute if only the load setting is changed between simulations.

Since the intended use for CPN modelling, like it is presented here, is analysis of NoC performance in an early stage of the design flow, the modelling effort is important to evaluate the usefulness of this method. This effort can be divided into two separate parts, the initial modelling of the NoC components and the combination of these component models to form a

complete NoC model. An overview of the time needs for CPN based modelling of a NoC is given in Table 2. Like the times needed for DSPN modelling these are related to the effort an experienced student has to spend. The time needed for modelling of the components varies from five minutes needed for the data *sink* model which only contains a single place and two transitions to six hours for more complex components like the *source* model or the *router* model.

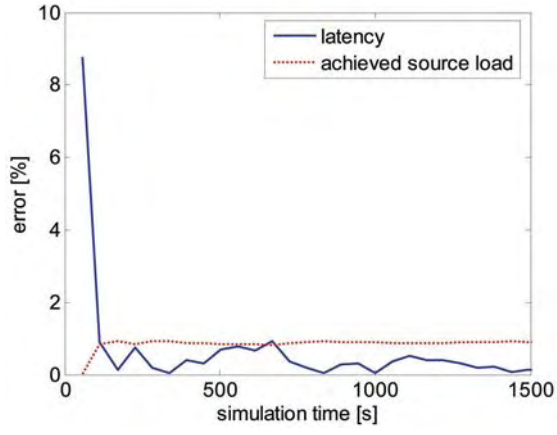


Fig. 27. Error of load and latency compared to emulation (requested load 20 %, 5x5 mesh)

Combining these components to the NoC model used in the experiments described in this section is done in approximately one hour. Exchanging single components, for example the *router* model when switching from experiment 1 to experiment 2, takes one minute but initiates a new syntax check that lasts for five minutes. Altering parameters such as the requested load setting is a matter of seconds but also requires a new partial syntax check of approximately two minutes.

modelling step	**effort**
modelling components	5 to 300 minutes
assembly of NoC model	60 minutes
complete syntax check	15 minutes
partial syntax check	< 5 minutes
simulation	5 minutes

Table 2. Time needs for CPN based modelling of a NoC

The results obtained and the precision achieved show that CPN based modelling of NoCs is an adequate approach for use in design space exploration of communication architectures. After initial modelling of NoC components only little time is needed to construct and modify a NoC model. The error of the results obtained by simulation of this model is small after a reasonably short simulation time. Furthermore, due to the modelling possibilities of CPNs complex components can be modelled with only a few places and transitions thus enabling the user to handle models of large NoCs with relative ease. These properties make CPN based modelling an attractive tool for pruning the design space by early elimination of NoC variants that do not provide the required performance with acceptable costs.

4. Conclusion and outlook

It is a key task of modern System-on-Chip (SoC) and Network-on-Chip (NoC) design to efficiently explore this design space regarding aspects like performance, flexibility and power consumption presumably in an early stage of the design flow in order to reduce design time and design costs.

In this chapter several examples for modelling of on-chip communication using Petri Net based modelling techniques have been presented. These examples include modelling of internal processor communication and modelling of inter-processor communication using a crossbar switch fabric. For these examples deterministic and stochastic Petri Nets have been applied as modelling technique. More complex NoC communication has been modelled applying Coloured Petri Nets. The results obtained with all of these models were compared to those calculated on an FPGA based emulator. In all presented experiments the performance measures derived using these models showed a good precision compared to the results acquired using the FPGA based emulator. Furthermore, the Petri Net based results could be derived in attractively short modelling times with only moderate effort.

Therefore, Petri Net based modelling of on-chip communication appears to be a very attractive approach to explore the design space of communication architectures in an early stage of the design process. DSPN based and CPN based modelling both provide specific advantages. DSPN models are suited for systems with moderate complexity such as communication systems with a small number of clients or bus based communication. The ease of modelling combined with the possibility of an analytical solution of the equations underlying the DSPN model provides a way to quickly obtain results. For more complex systems including a lot of data and complex functionalities, for example the addressing scheme and the routing algorithm in a NoC, CPN models are more adequate. DSPN based modelling of such systems is not as efficient since DSPNs do not provide a means of modelling data structures. As CPNs include data structures and allow to model complex behaviour in form of colorsets and transfer functions, CPN based modelling is well suited to analyze complex on-chip communication systems.

Current topics in the field of NoC communication modelling to be addressed with Petri Net based methods are locating hotspots, analyzing quality-of-service aspects (data integrity, guaranteed service, etc.) and complex adaptive routing algorithms (incl. the checking of absence of deadlocks).

5. References

Ajmone Marslan, M.; Chiola, G. (1987). On Petri Nets with Deterministic and Exponentially Distributed Firing Times, in G. Rozenberg (Ed.) *Advances in Petri Nets* 1986, Lecture Notes in Computer Science, Vol. 266, Springer, pp. 146-161

Altera (2001). Nios Embedded Processor Software Development Reference Manual.

Altera (2007). http://www.altera.com

Avalon (2007). http://www.altera.com/literature/manual/mnl_avalon_spec.pdf Bus specification manual.

Benini, L. & de Micheli, G. (2002). Networks on Chips: A New SoC Paradigm, Computer, Vol. 35, Iss. 1, pp. 70-78, January 2002, ISSN 0018-9162

Bjerregaard, T. & Mahadevan, S. (2006). A Survey of Research and Practices of Network-on-Chip, ACM Computing Surveys, Vol. 38, article 1, March 2006, ISSN 0360-0300

Blume, H.; Feldkämper, H.; Noll, T. G. (2005). Model-based Exploration of the Design Space for Heterogeneous Systems-on-Chip, *Journal of VLSI-Signal Processing*, Vol. 40, Nr. 1, May 2005, pp. 19-34

Blume, H.; von Sydow, T.; Becker, D. Noll, T. G. (2007). Application of Deterministic and Stochastic Petri Nets for Performance Modelling of NoC Architectures, *Journal of Systems Architecture*, Vol. 53, Issue 8, 2007, pp. 466-476

Blume, H.; von Sydow, T.; Noll, T. G. (2006). A Case Study for the Application of Deterministic and Stochastic Petri Nets in the SoC Communication Domain, *Journal of VLSI Signal Processing 2006*, Vol. 43, Nr. 2-3, June 2006, pp. 223-233

Ciardo, G.; Cherkasova, L.; Kotov, V.; Rokicki, T. (1995). Modelling a scalable high-speed interconnect with stochastic Petri Nets, in: *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models PNPM'95* October 03–06, Durham, North Carolina, USA, pp. 83–94.

DSPNexpress (2003). http://www.dspnexpress.de

Duato, J.; Yalamanchili, S. & Ni, L. (2003). *Interconnection Networks – An Engineering Approach*, Morgan Kaufmann, ISBN 0818678003 , San Francisco

Jensen, K. (1980). Net Models in System Development, PhD thesis, Aarhus University

Kleinrock, L. (1975). *Queueing Systems* – Vol. 1: Theory, John Wiley and sons

Kogel, T.; Doerper, M. et al. (2003). A modular simulation framework for architectural exploration of On-Chip interconnection networks, *CODES + ISSS*, October 2003.

Lahiri, K.; Raghunathan, A.; Dey, S. (2001). System-level performance analysis for designing On-Chip communication architectures, *IEEE Transactions on CAD of Integrated Circuits and Systems*, June 2001.

Lindemann, C. (1998). *Performance Modelling with Deterministic and Stochastic Petri Nets*, John Wiley and sons, ISBN 0471976466, Berlin

Madsen, J.; Mahadevan, S.; Virk, K. (2004). Network-centric systemlevel model for multiprocessor SoC simulation, in: J. Nurmi et al. (Eds.), *Interconnect Centric Design for Advanced SoC and NoC*, Kluwer Academic Publishers

Mickle, M. H. (1998). Transient and steady-state performance modelling of parallel processors, *Applied Mathematical Modelling* 22 (7) (1998) 533–543

Moore, G. (1965). Cramming more components onto integrated circuits, *Electronics*, Volume 38, Number 8, April 19, 1965

Neuenhahn, M.; Blume, H.; Noll, T. G. (2006). Quantitative analysis of network topologies for NoC-architectures on an FPGA-based emulator, .*Proceedings of the URSI Advances in Radio Science - Kleinheubacher Berichte*, Miltenberg, September 2006

Petri Nets World (2007). http://www.informatik.uni-hamburg.de/TGI/PetriNets/

Plosila, J.; Seceleanu, T.; Sere, K. (2004). Formal communication modelling and refinement, in: J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch (Eds.), *Interconnect Centric Design for Advanced SoC and NoC*, Kluwer Academic Publishers

Ratzer, A. V.; Wells, L.; Lassen, H. M.; Laursen, M.; Qvortrup, J. F.; Stissing, M. S.; Westergaard, M.; Christensen, S. & Jensen, K. (2006). CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets, *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN) 2003*, pp. 450-462, ISSN 0302-9743, Eindhoven, June 2003, Springer Verlag, Berlin

Sonntag, S.; Gries, M.; Sauer, C. (2005). SystemQ: A Queuing-Based Approach to Architecture Performance Evaluation with SystemC, *Proceedings of the SAMOS V Workshop*, Samos, Greece, July 18-20 2005, LNCS 3553, ISBN 354026969, pp. 434-444

SOPC (2007). http://www.altera.com/products/software/products/sopc/sop-index.html

Texas Instruments (2007). http://www.ti.com

Zaitsev, D. A. (2004). An Evaluation of Network Response Time using a Coloured Petri Net Model of Switched LAN In: K. Jensen (ed.): *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, October 2004, Department of Computer Science, University of Aarhus, PB-570, 157-166.

4

An Inter-Working Petri Net Model between SIMPLE and IMPS for XDM Service

Jianxin Liao[1], Yuting Zhang[1, 2] and Xiaomin Zhu[1]
[1]*State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications*
[2]*China Mobile Research Institute*
China

1. Introduction

The XDM (XML Document Management) (Open Mobile Alliance, 2006a ; Open Mobile Alliance, 2006c; Open Mobile Alliance, 2006h ; Open Mobile Alliance, 2006l) service is more and more attractive in the age of regarding the user as the communication center. XDM which was ever named GM (Group Management) comes out with the Instant Messaging (IM) service, and manages the personal information, such as contact list, which is the necessary information that IM service needs (Rosenberg, 2005). With the development of XDM service, the information which the XDM server stores will be richer and richer, for example, the personal profile and group information are added to the XDM service. Then the XDM service turns into one of the supporting services from a simple service which just provides the contact for the user. As one of the supporting services, the XDM provides the personal information for the other services which need them, for example, providing the user's personal information for the IM service or Presence service. At present, there are many different appellations of XDM in different standards. Many research institutes and companies have presented their implementations for the XDM service, but the implementations are different and incompatible, so it is hard to inter-work between them. However the inter-working for the services based on XDM service is desired intensively, such as IM, Presence service. There are three main international standards for the XDM service and its relative services: SIMPLE (Session Initiation Protocol for Instant Message and Presence Leveraging Extensions) {1}, IMPS (Instant Messaging and Presence Services) {2} and XMPP (Extensible Messaging and Presence Protocol) (P. Saint-Andre, 2004a ; P. Saint-Andre, 2004b). IMPS has been deployed in many systems for its better maturity, but SIMPLE is more suitable for being deployed in IMS (Internet Protocol Multimedia Subsystem) network, so inter-working between them is becoming a research hotspot in the value-added service field, but recently the research on the topic is just in the initial stage. OMA (Open Mobile Alliance) just proposed a simple Architectural Model (Open Mobile Alliance, 2005a; Open Mobile Alliance, 2005b) for the inter-working between SIMPLE and IMPS, but the architecture can not perform the inter-working for the XDM service. Based on the research on the difference between SIMPLE and IMPS and the OMA's inter-working Architectural Model, we (with the State Key Lab of Networking and Switching Technology of Beijing University of Posts and Telecommunications, and Research Institute of China Mobile) have proposed a bi-directional protocol mapping for use to enable the exchange of XDM

information and an Enhanced Architectural Model to perform the inter-working functions which can not be completed by the current OMA Architectural Model. On the other hand, there is no any other research institute studying the XDM inter-working with the Protocol Conversion Methodology (Green, 1988; Zhu, 2006b; Zhu, 2007). In this chapter, according to the method proposed by the Protocol Conversion Methodology, the XDM inter-working model based on Petri Nets {3} is set up to verify the mapping and the Enhanced Architectural Model by a new coupling criteria of Petri net model. After the strict mathematical analysis and verification for the model, which prove that the model meets all properties of a correct Petri net model, the mapping and the Enhanced Architectural Model are proved to be reasonable and viable, and the probable exceptions in the inter-working can be found and excluded. During the modeling experiences of the inter-working with Petri Nets, some methods for solving the conflict of a Petri Net are proposed, which enriches the application of Petri Nets for the protocol conversion. As the concept of XDM is almost same among different standards, the inter-working Petri net model can provide an applicable reference for the inter-working between other standards.

2. SIMPLE and IMPS

2.1 SIMPLE

IETF (the Internet Engineering Task Force) first set up the basic models (Day, 2000a; Day 2000b) for Presence and Instant Messaging, then a working group called SIMPLE was founded to focus on the application of the SIP (Session Initiation Protocol) (Rosenberg, 2002) to the suite of services collectively known as Instant Messaging and Presence (IMP) {1}. 3GPP and 3GPP2 adopt SIMPLE as their basic standard and specify the practical implementations of SIMPE specifications for the Presence, Group and IM service in IMS and MMD (MultiMedia Domain) respectively (3GPP, 2002-2005f; 3GPP2, 2002-2006). With the expanding of the influence of SIMPLE standards, OMA also adopts it as its basic standard and has set up workgroups to create application level specifications for this standard. SIMPLE was divided into three services in OMA: Presence service, XDM (XML Document Management) service and IM service, and these three services would be in progress independently.

2.2 IMPS

IMPS, which is also based on the basic models set up by IETF, was designed for exchanging instant messages and Presence Information not only between mobile devices but also between mobile and fixed devices, which includes four primary features {2} (Open Mobile Alliance , 2007a-2007m) : Presence, Instant Messaging, Groups and Shared Content.

2.3 XDM service

In SIMPLE, XDM service provides the Personal Information (Shared Profile), Shared List and Shared Group for the users and the other services (Open Mobile Alliance, 2006a-2006l). There is almost the same information in IMPS. The Personal Information belongs to the common features in IMPS, which includes two parts: Private Profile and Public Profile. The Contact List (i.e. Shared List) belongs to the Presence features in IMPS. Group is a set of users, which is the same feature in SIMPLE and IMPS, and the participant can see the other participants in the same Group. The Group participant can have a group conversation (Instant Message conversation or voice conversation) through the Group information. Besides the conversation between the participants, the communication request initiated by

one of the participants can be received by the other participants. Why there is a "Shared" in front of the "List" and "Group" in SIMPLE is just because the List and Group in SIMPLE can be referenced by the other service, but the List and Group in IMPS can't, which are only used for Instant Messaging.

As there is almost the same information within SIMPLE and IMPS, it is possible for them to inter-work, a bi-directional protocol mapping for use to enable the exchange of XDM information is proposed to perform the inter-working functions (Zhang, 2007) (also see the mapping described in general in Section 3). Here we also use the XDM service to describe the corresponding service in IMPS.

2.4 Inter-working between SIMPLE and IMPS

At present, only OMA had made some contributions which just focused on a simple Architectural Model on the inter-working subject (Open Mobile Alliance, 2005a; Open Mobile Alliance, 2005b). State Key Laboratory of Networking and Switching Technology of Beijing University of Posts and Telecommunications and Research Institute of China Mobile have paid much attention to the inter-working issues and spent a lot of care on researching them. On the study of OMA's inter-working Architectural Model and the gaps between SIMPLE and IMPS standards, we found the OMA's Architectural Model can only realize the inter-working of Presence and Instant Messaging, and the inter-working for the other important service named XDM service can not be completed by the current OMA Architectural Model. Because the XDM service is going to be more and more important as it has become one of the basic supporting service, if the inter-working for XDM service has been realized, the XDM service and the services supported by XDM service would be more attractive to the user. We proposed an Enhanced Architectural Model for the inter-working (Zhang, 2007), as shown in Figure 1.

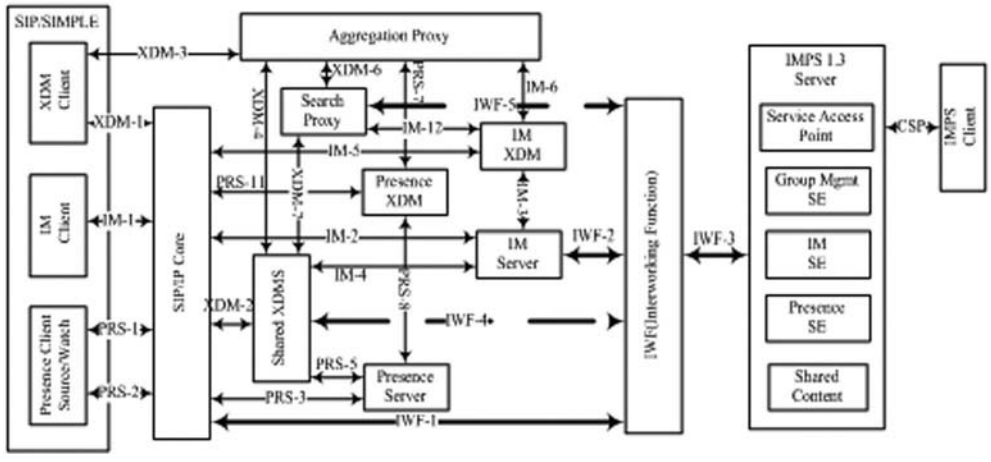


Fig. 1. The Enhanced IWF Architectural Model

In this Model, we set up a new reference point IWF-4 to support the communication between the Shared XDMS and IWF, set up a new reference point IWF-5 to support the searching inter-working between IWF and Search Proxy which has been defined to search XML documents stored in any XDM Servers in OMA SIMPLE XDM 2.0. The IWF-4 reference point based on XCAP (Extensible Markup Language Configuration Access Protocol)

(Rosenberg, 2006) performs the inter-working functions for the Group and Shared Profile features:

1. Retrieve the Public Profile and Group information from the Shared XDMS to the IWF Server;
2. Retrieve the Shared Profile and Shared Group information from the IWF Server to the Shared XDMS.
3. Add/Remove Group member from the Shared XDMS to the IWF Server;
4. Add/Remove Group member from the IWF Server to the Shared XDMS;
5. Leave/Join a Group from the Shared XDMS to the IWF Server;
6. Leave/Join a Group from the IWF Server to the Shared XDMS.

The IWF-5 reference point based on XCAP performs the inter-working functions for searching:

1. Search for users/groups from the Search Proxy to the IWF Server;
2. Search for users/groups from the IWF Server to the Search Proxy.

Besides, the functions of the IWF-1 reference point should be enhanced:

1. Subscription of changes in state of Group from the Shared XDMS to the IWF;
2. Notification of changes in state of Group from the IWF to the Shared XDMS.

The functions of the IWF-3 reference point should also be enhanced:

1. Retrieve the Public Profile and Group information from the IWF Server to the IMPS Server;
2. Retrieve the Shared Profile and Shared Group information from the IMPS Server to the IWF Server;
3. Subscription of changes in properties of group in the IMPS Server;
4. Notification of changes in group properties;
5. Add/Remove Group member from the IMPS Server to the IWF Server;
6. Add/Remove Group member from the IWF Server to the IMPS Server;
7. Leave/Join a Group from the IMPS Server to the IWF Server;
8. Leave/Join a Group from the IWF Server to the IMPS Server;
9. Search for users/groups from the IWF Server to the IMPS Server;
10. Search for users/groups from the IMPS Server to the IWF Server.

The functions of the IWF should be enhanced too:

1. Protocol conversions between XCAP and SSP;
2. Information Mappings and information format conversions between XDM information and IMPS corresponding information, such as Group information, Public Profile;
3. Operation mappings between XDM service and IMPS corresponding service in the new references.

After the IWF-4 reference point is added, the IMPS users can use the SIMPLE Group for communication and can retrieve the information of the Group participant who is the SIMPLE user whenever they want, as long as the information has been authorized by the SIMPLE user. Also, the SIMPLE users can do the same thing. After the IWF-5 reference point is added, the users can search for the users/groups not only in their service system, but also outside of their service system.

3. Petri net modeling

In order to verify the mapping and the Enhanced Architectural Model, and exclude the probable exceptions in the inter-working, we should set up some means to verify and

analyze it.

The inter-working between SIMPLE and IMPS is mainly involved in the conversion between the protocols, and how to realize the inter-working is involved in the Protocol Conversion Methodology. As the Petri Nets, which is based on a set of strict mathematical theories, is very useful for the verification and analysis of the inter-working model (Zhu, 2007), this chapter adopts Petri Nets theory to set up inter-working model to analyze and verify the Enhanced Architectural Model and the related inter-working mapping.

We adopt the protocol conversion methodology of the Petri net model which is proposed by (Green, 1988; Zhu, 2006b) to build the Petri net model for XDM Service in the following section.

3.1 Atomic protocol functions of different reference points

The inter-working of the XDM service is mainly involved in the reference points IWF-1, IWF-3, IWF-4 and IWF-5. As described in (Zhang, 2007), IWF-1 includes the following SIP methods and their corresponding atomic protocol functions, as shown in Table 1.

Atomic Protocol Functions	SIP Request Methods
Subscribe Group Change	SUBSCRIBE (expires>0)
Notify Group Change	NOTIFY
Unsubscribe Group Change	SUBSCRIBE (expires=0)

Table 1. The atomic protocol functions in IWF-1 for XDM service

The methods and their atomic protocol functions in IWF-3 are shown in Table 2.

Atomic Protocol Functions	SSP Methods
Retrieve user's profile	GetPublicProfileRequest、GetPublicProfileResponse
Retrieve the Group Information	GetGroupPropsRequest、GetGroupPropsResponse
Retrieve the joined member list of a group	GetJoinedMemberRequest、GetJoinedMemberResponse
Retrieve the member list of a group	GetGroupMemberRequest、GetGroupMemberResponse
Join Group	JoinGroupRequest、JoinGroupResponse
Leave Group	LeaveGroupRequest、LeaveGroupIndication
Server Initiated Leave Group	LeaveGroupIndication、Status
Add Group Member	AddGroupMemberRequest、Status
Remove Group Member	RemoveGroupMemberRequest、Status
Subscribe Group Change	SubscribeGroupChangeRequest、Status
Notify Group Change	GroupChangeNotice、Status
Unsubscribe Group Change	UnsubscribeGroupChangeRequest、Status
Search for Users or Groups	SearchRequest、SearchResponse

Table 2. The atomic protocol functions in IWF-3 for XDM service

The methods and their atomic protocol functions in IWF-4 are shown in Table 3.

Atomic Protocol Functions	XCAP Request Methods
Retrieve user's profile	XCAP GET
Retrieve the Group Information	XCAP GET
Retrieve the member list of a group	XCAP GET
Join Group	XCAP PUT
Leave Group	XCAP DELETE
Add Group Member	XCAP PUT
Remove Group Member	XCAP DELETE

Table 3. The atomic protocol functions in IWF-4 for XDM service

Although Table 3 has many request methods of XCAP GET, XCAP PUT and XCAP DELETE, the URI (Uniform Resource Identifier) of every XCAP request method is different, so the request methods listed in the table indicates different concrete requests.
The methods and their atomic protocol functions in IWF-5 are shown in Table 4.

Atomic Protocol Functions	XCAP Request Methods
Search for Users or Groups	XCAP GET

Table 4. The atomic protocol functions in IWF-5 for XDM service
Although the methods of XDM service involved in different reference points only include the methods listed from Table 1 to Table 4, in the process of real service execution, the other XDM service methods may invoke some methods described above, for example, the change of the attributes of Group Information will cause the notification of the Group Information in SIMPLE, so will it in IMPS.

3.2 Common subset of atomic protocol functions

The corresponding relation of above atomic protocol functions: the atomic protocol functions in Table 1, Table 3 and Table 4 are corresponding to the atomic protocol functions in Table 2 respectively. We can see that there are some atomic protocol functions in Table 2 which can not be mapped to the other atomic protocol functions in the other tables. That is to say, some atomic protocol functions in IWF-3 can not be corresponding to the atomic protocol functions in the other reference points, i.e. "Retrieve the joined member list of a group" and "Server Initiated Leave Group" can not find their corresponding atomic protocol functions in Table 3. Except the two atomic protocol functions, the other atomic protocol functions can find their corresponding atomic protocol functions very well.

3.3 Decide whether the common subset is satisfied with the inter-working requirement

From the common subset of the atomic protocol functions, it is basically satisfied with the inter-working requirement, but not completely satisfied, so it is soft mismatch (Green, 1988) in the protocol conversion, and we should carry out the protocol complementing for the requirement.

3.4 Protocol complementing

As the common subset is not completely satisfied with the inter-working requirement, we should complement some atomic protocol functions in IWF-4, as shown in Table 5. The XCAP request methods added are based on the existed XCAP request methods. These XCAP request methods have different request URIs.

Atomic Protocol Functions	XCAP Request Methods
Retrieve the joined member list of a group	XCAP GET
Server Initiated Leave Group	XCAP DELETE

Table 5. The complemented atomic protocol functions in IWF-4 for XDM service

3.5 The common subset of atomic protocol functions after protocol complementing

After the protocol complementing, we get the whole common subset of atomic protocol functions which the service implementation needs. Table 6 shows mappings of atomic protocol functions between the IWF-1 and IWF-3.

Atomic protocol functions on IWF-1	Atomic protocol functions on IWF-3
Subscribe Group Change	Subscribe Group Change
Notify Group Change	Notify Group Change
Unsubscribe Group Change	Unsubscribe Group Change

Table 6. Semantic Mapping Relationship of Atomic Protocol Functions between IWF-1 and IWF-3

The mapping relationship of atomic protocol functions between IWF-4 and IWF-3 is shown in Table 7.

Atomic protocol functions on IWF-4	Atomic protocol functions on IWF-3
Retrieve user's profile	Retrieve user's profile
Retrieve the Group Information	Retrieve the Group Information
Retrieve the joined member list of a group	Retrieve the joined member list of a group
Retrieve the member list of a group	Retrieve the member list of a group
Join Group	Join Group
Leave Group	Leave Group
Server Initiated Leave Group	Server Initiated Leave Group
Add Group Member	Add Group Member
Remove Group Member	Remove Group Member

Table 7. Semantic Mapping Relationship of Atomic Protocol Functions between IWF-4 and IWF-3

The mapping relationship of atomic protocol functions between IWF-5 and IWF-3 is shown in Table 8.

Atomic protocol functions on IWF-5	Atomic protocol functions on IWF-3
Search for Users or Groups	Search for Users or Groups

Table 8. Semantic Mapping Relationship of Atomic Protocol Functions between IWF-5 and IWF-3

It should be based on the mapping relationship of XDM information and corresponding

methods described in (Zhang, 2007) to realize the conversion of the XDM information and methods, when we implement the inter-working for the information and methods in the common subset.

3.6 Confirmation of the conversion way

Because it involves the SIP, XCAP and SSP protocols in the process of the protocol conversion in the IWF, and it needn't carry out the conversions between multiple protocols, we decide to convert the protocols directly. Based on the description above, now we can set up a Petri net model to verify the mapping and the Enhanced Architectural Model with strict mathematical analysis and verification in order to find and exclude the probable exceptions in the inter-working.

3.7 Petri net model

Only the atomic protocol functions listed in Table 6 are totally related. Some atomic protocol functions listed in Table 7 often indicate the separate service methods, and always have nothing to do with the other atomic protocol functions listed in the same table, for example, "Retrieve user's profile" has distant relationship to the "Retrieve the Group Information", "Retrieve the Group Information" has distant relationship to the "Retrieve the member list of a group". Someone joining a Group or adding a group member is the precondition for the atomic protocol functions listed in Table 6 to be effective, and it can also make the group to be changed. In the modeling procedure described in this section, we first model the atomic protocol functions listed in Table 6, then model the representative atomic protocol functions listed in Table 7 and Table 8, at last we couple the related Petri net models for the different reference points to an integrated Petri net model by a new coupling criteria.
Comparing multiple computer-aided Petri nets tools, we decide to adopt Visual Object Net++ {4} to construct the Petri net model for the XDM service.

3.7.1 Subscribing group change

According to the service flow and the mapping described in (Zhang, 2007), we construct the MSC (Message Sequence Chart) of Subscribing Group Change, then we educe the corresponding Petri net model, as shown in Figure 2.
Figure 2 is composed of two Sub-Figures and several places and transitions that couple the two Sub-Figures. The symbol "+" before every transition means that the corresponding transition has received a message from the related arc, while the symbol "-" means that the corresponding transition has sent out a message.
Sub-Figure 2(a) shows the Petri net model for Subscribing Group Change in IWF-1. The transitions and their possible occurring sequences, which are within three round-corner rectangles in Sub-Figure 2(a) represent the following atomic protocol functions: Subscribe Group Change, Notify Group Change and Unsubscribe Group Change. The six places (P6, P7, P8, P10, P11, P12) in Sub-Figure 2(a) represent the state elements in external channel between Shared XDMS and IWF. Sub-Figure 2(b) shows the Petri net model for Subscribing Group Change in IWF-3. The transitions and their possible occurring sequences, which are within three round-corner rectangles in Sub-Figure 2(b) represent the following atomic protocol functions: Subscribe Group Change, Notify Group Change and Unsubscribe Group Change. The six places (P29, P30, P31, P32, P33, P34) in Sub-Figure 2(b) represent the state elements in external channel between IWF and IMPS server.
According to Table 6 and the coupling criteria proposed in (Zhu, 2007), we couple the Petri net model in Sub-Figure 2(a) and the Petri net model in Sub-Figure 2(b) into an integrated

Petri net model for Subscribing Group Change, as shown in Figure 2. The six places (P18, P19, P20, P21, P22, P23) represent the state elements in internal channel in IWF.

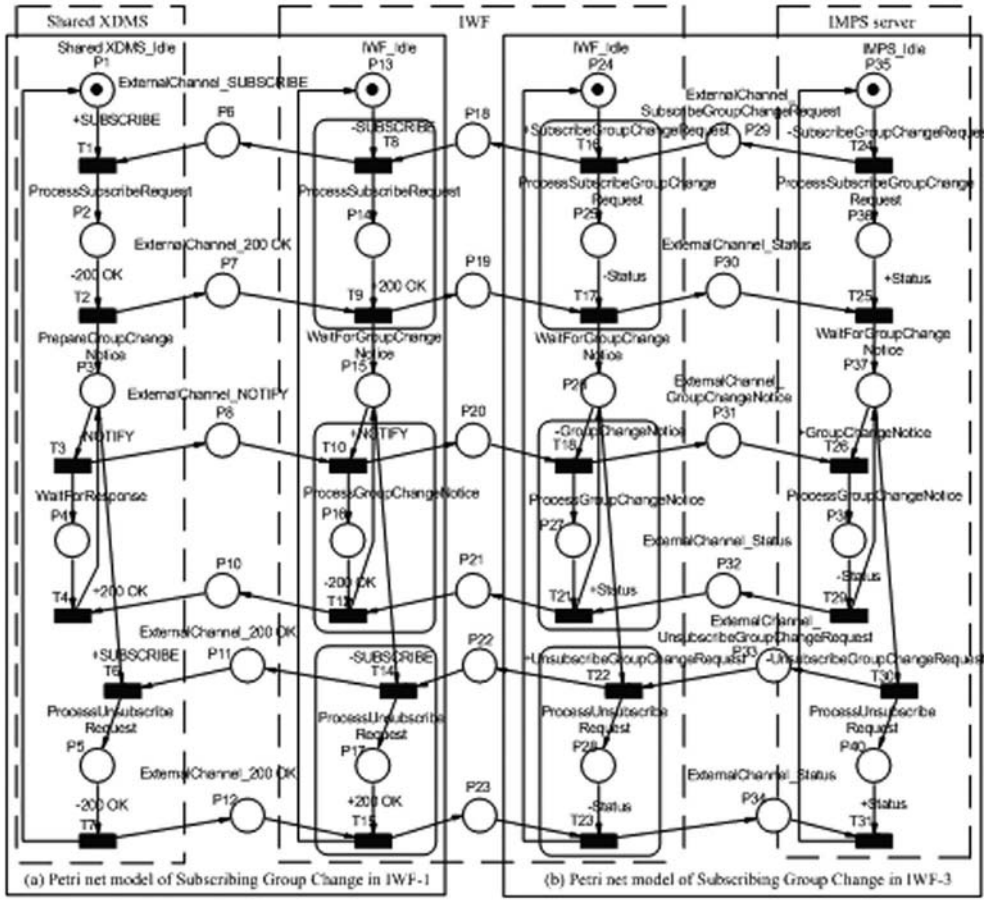


Fig. 2. The Petri net model of Subscribing Group Change

This model describes the case of Subscribing Group State on the condition that the subscribing request is initiated from IMPS server. The case which indicates the subscribing request initiated from Shared XDMS is not in the same session with the case showed in Figure 2, and the groups which the two cases represent are different, so the Petri net model which describes the case on the condition that the subscribing request is initiated from Shared XDMS should be set up in another model. In the construction process, the corresponding places, transitions and related arcs should be constructed in a reverse direction from the model described in Figure 2, according to real condition of the service implementation, not simply constructed directly from the above model in the reverse direction. The two models are symmetrical in some degree.

Because the model shown in Figure 2 has unexpected conflicts and deadlocks (see the analysis of the model in the Section 4), it represents that the existing mapping may have

some exceptions or unconsidered issues. In order to make the conflicts and deadlocks free, we construct the modified Petri net models, as shown in Sub-Figure 4(b) and Sub-Figure 4(e).

Based on the model shown in Figure 2, we add a series of places and transitions to build the model shown in Sub-Figure 4(b) and Sub-Figure 4(e), for example, add P9 in the external channel between Shared XDMS and IWF, add T11 and T12 in the IWF. Some places and transitions added represent the new mappings, such as T5, P9 and T12 represent a new mapping, i.e. the 487 Response of SIP sent from IWF to Shared XDMS, while some represent the new state, such as P39 represents the state in which the system only receives the notification of group state change (i.e. it doesn't receive the notification of unsubscribing group change).

3.7.2 Join group

Join Group is one of the representative atomic protocol functions listed in Table 7, and its completion is the precondition of the cases listed in Table 6. The implementation of Add Group Member is similar to that of Join Group. Leave Group, Server Initiated Leave Group and Remove Group Member are almost the reverse operation of Join Group (or Add Group Member). Except those atomic protocol functions, the other atomic protocol functions are all independent. The cases described by the atomic protocol functions listed in Table 8 are also independent, so this section takes an example of Join Group to construct the corresponding Petri net model, as shown in Figure 3.

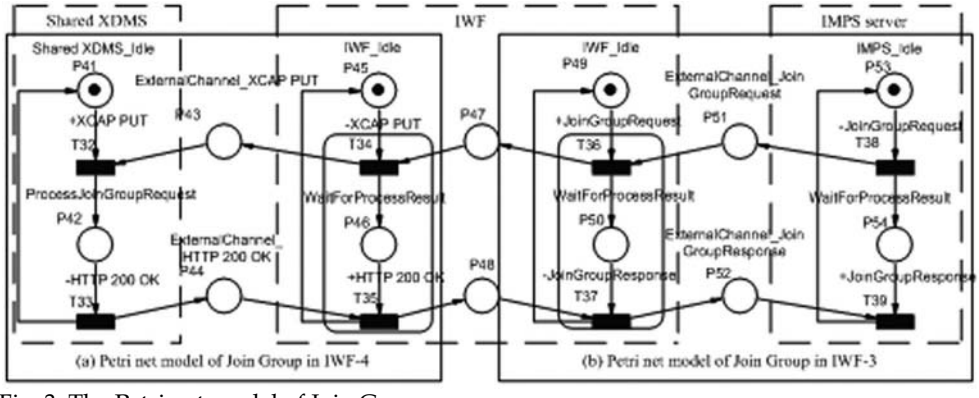


Fig. 3. The Petri net model of Join Group

Figure 3 is composed of two Sub-Figures and several places and transitions that couple the two Sub-Figures. Sub-Figure 3(a) shows the Petri net model for Join Group in IWF-4. The transitions and their possible occurring sequences, which are within the round-corner rectangle in Sub-Figure 3(a) represent the atomic protocol function: Join Group. The two places (P43, P44) in Sub-Figure 3(a) represent the state elements in external channel between Shared XDMS and IWF. Sub-Figure 3(b) shows the Petri net model for Join Group in IWF-3. The transitions and their possible occurring sequences, which are within the round-corner rectangle in Sub-Figure 3(b) represent the atomic protocol function: Join Group. The two places (P51, P52) in Sub-Figure 3(b) represent the state elements in external channel between IWF and IMPS server.

According to Table 7 and the coupling criteria proposed in (Zhu, 2007), we couple the Petri net model in Sub-Figure 3(a) and the Petri net model in Sub-Figure 3(b) into an integrated Petri net model for Join Group, as shown in Figure 3. The two places (P47, P48) represent the state elements in internal channel in IWF.
This model describes the case of Join Group on the condition that the joining request is initiated from IMPS server, just like the analysis described in Section 3.7.1, the Petri net model which describes the case on the condition that the joining request is initiated from Shared XDMS should be set up in another model.

3.7.3 The coupled model and the coupling criteria

The message flows between IWF-1 and IWF-3 need cooperation with the message flows between IWF-4 and IWF-3, so the Petri net models described above should be further coupled together. This chapter takes an example of coupling the Join Group, Subscribing Group Change and Leave Group into an integrated Petri net model to show how an integrated case of XDM service is completed. Other small cases represented by other atomic protocol functions can replace one of the small cases represented by Join Group, Subscribing Group Change or Leave Group in order to model other integrated case. For example, the case of Add Group Member can replace the case of Join Group, the case of Retrieve the Member List of a Group can replace the case of Subscribe Group Change, the difference between Retrieve the Member List of a Group and Subscribe Group Change is that the two atomic protocol functions are involved in different relationships between different reference points, so in order to simplify the coupled model, the service state will enter the case of Subscribe Group Change directly after the user has joined the group in the coupled model. When a user unsubscribe the change of a group, the user may leave the group, or be removed by the group manager, in this chapter, we only consider the case of user leaving group for the convenient modeling. The coupled model is shown in Figure 4. Compared to Figure 2 and Figure 3, the same part in Figure 4 is indicated as suspension points. The Petri net model of Leave Group is almost like the Petri net model of Join Group, so we don't describe the Petri net model separately, but describe it in the coupled model, as shown in the last part of Figure 4.
The coupling criteria proposed in (Zhu, 2007), is referenced by us when we commence coupling. In the criteria, the coupling of service parallel execution is realized by adding new places and changing the direction of directed arc from the places. We don't adopt the criteria completely, but create a new criteria:

- Adding new places and transitions, changing the direction of directed arc at the same time, such as P55, T40, P70, T49;
- Besides the above rule, the value of the token in the places connected with the coupling transitions in the original model should be set to zero, and the direction of the corresponding directed arc should be changed, in order to ensure the service is executed naturally, such that the values of the tokens in P35 and P68 are set to zero, the direction of T39 pointing to P53 is changed to point to P55.

The coupling of service serial execution is realized by the new criteria. In the criteria, the added places, transitions and the directed arcs connected with these places and transitions are the coupling points. These coupling points are those parts in which the messages in different reference points should be cooperated, which has special state and different events, so the coupling points should be paid more attention to when we implement the system. The Petri net model coupled by the new criteria meets all properties of a correct Petri net model, please see the analysis of the model in the Section 4.

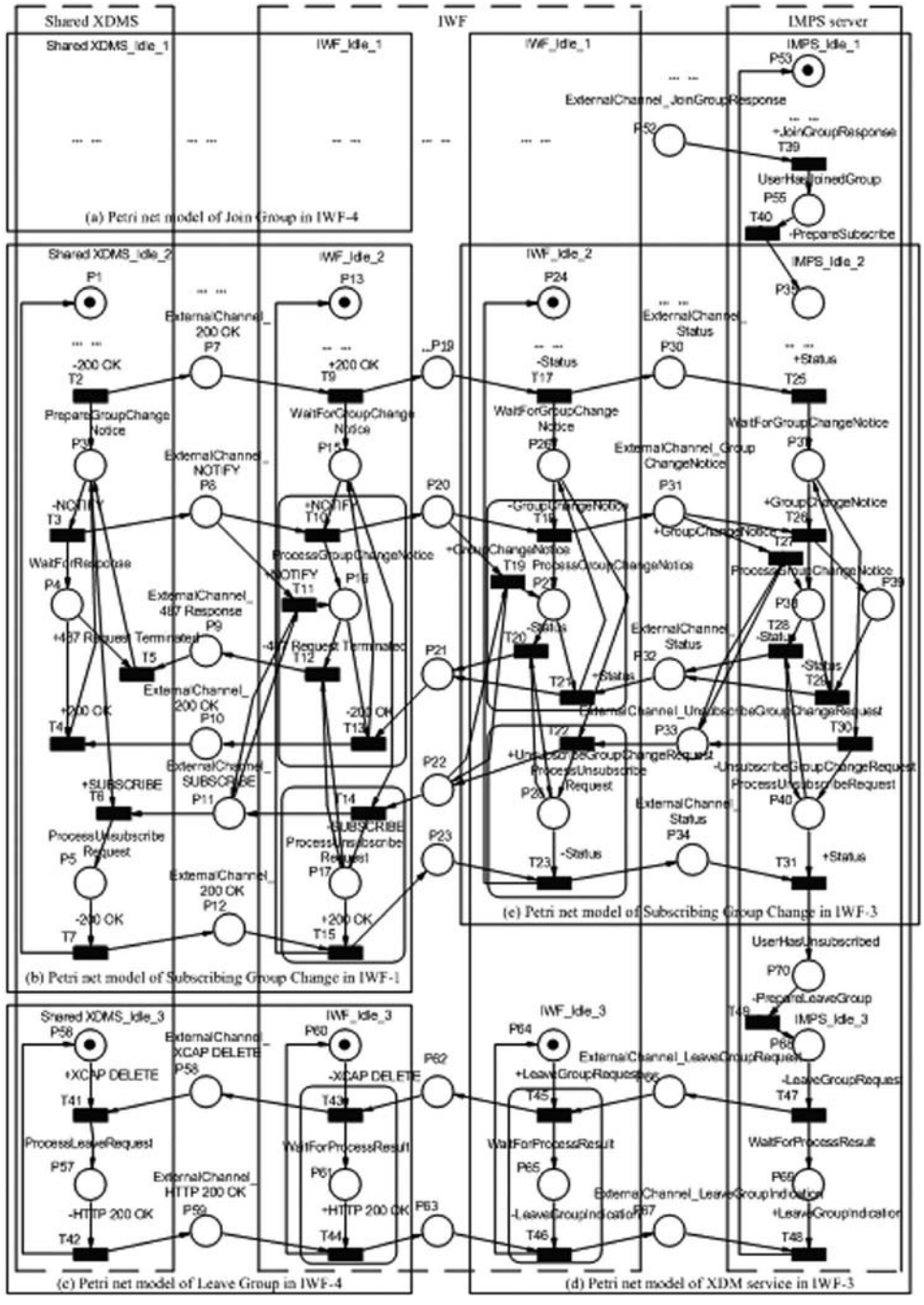


Fig. 4. The coupled Petri net model for XDM service

4. Analysis of the model

We analyze the properties of the Petri net model by combined analysis method of simulation analysis, reachability analysis, invariant analysis. The simulation analysis is completed by Visual Object NET++ and PIPE2 (Platform Independent Petri Net Editor 2) {5}, and the invariant analysis is completed by PIPE2. A correct Petri net model for protocol conversion should have the following attributes: Boundedness, Conflict freeness, Contact freeness, Deadlock freeness, Livelock freeness, Resetability, S-invariant (Zhu, 2007).

The coupled Petri net model has remained the properties of each original model, so we only analyze the properties of the coupled Petri net model from which the properties of each original model can be known, for reducing the length of this chapter.

We make the simulation analysis by Visual Object Net++ and PIPE2 for Figure 4. From the analysis of structural properties of the model, the model is not a pure net, not a simple net either. But from the analysis of state space of the model, the model is safe, i.e. 1-Boundedness, live, Contact freeness, Deadlock freeness and Livelock freeness.

There are four possible conflict groups: {T26, T30} with the reachable marking M^1 (P4=P16=P27=P31=P37=1, the token values of the rest places are 0), {T18, T22} with the reachable marking M^2 (P4=P16=P20=P26=P33=P40=1, the token values of the rest places are 0), {T10, T14} with the reachable marking M^3 (P4=P8=P15=P22=P28=P40=1, the token values of the rest places are 0), {T3, T6} with the reachable marking M^4 (P3=P11=P17=P28=P40=1, the token values of the rest places are 0). This model is coupled with the modified Petri net model from the original model shown in Figure 2, after resolving the problems found from the original model. In the original model, there are four conflict groups described above, but in any of the markings of M^2 and M^3, the happening of any of the transitions in the conflict group will cause the deadlock of the system, also, in the marking of M^1, the happening of T30 will cause the deadlock of the system, in the marking of M^4, the happening of T3 will cause the deadlock of the system. In fact, it indicates that there are some exceptions in the original mapping, for example, in the marking M^4, when Shared XDMS has just sent the NOTIFY, i.e. T3 has just happened, but at the same time, IWF has just received the UnsubscribeGroupChangeRequest from IMPS server and sent out SIP SUBSCRIBE converted from the request for unsubscribing, i.e. T14 has also happened, it means that IWF has stopped process the SIP NOTIFY when IWF has just received NOTIFY, because it has received the unsubscribing request from IMPS server, and at this time, it is not good for IWF to convert NOTIFY to GroupChangeNotice which will be sent to IMPS server in the next step, or throw away NOTIFY (if NOTIFY is thrown away, it betrays the principle of SIP), so the system is "dead". In order to resolve the problem, we extend the message flow in the mapping. When IWF has received NOTIFY and unsubscribing request, IWF sends the 487 Response (Request Terminated) to Shared XDMS, so we add T5, T12 and P9 to mark this response. In order to distinguish the NOTIFY received normally from the NOTIFY received abnormally (i.e. IWF has received the unsubscribing request and sent it out when IWF has received NOTIFY), T11 is added to represent receiving NOTIFY abnormally. After the addition has been done, the deadlock brought from M4 will never happen, and the conflict brought from M4 will become the "untrue" conflict. When T3 has happened, it is sure for T6 to have a chance to happen after some transitions have happened in the Petri net model, so we deem the conflicts are not the actual conflicts. It meets the principles of a correct Petri net model, for the Petri net model can well guide the development of a real system and well simulate the possible exceptions in the real environment after the Petri net model has been modified by the above way, which is consistent with the real application environment (it is possible for the request to be delayed for some reasons). It shows the strong capability of

strict mathematical analysis of Petri net in another way, which can expose the possible problems before system implementation by the properties of deadlock, conflict and so on, and can help us to resolve these possible problems and decrease the errors in system implementation, as the resolving way of Petri net. If we want to resolve these conflicts further, we can treat T6 and T3 as immediate transition and timed transition, or give every transition a different execution probability, which are not shown in this chapter for model simplicity reason. There are similar problems in the marking of M^1, M^2 and M^3, but in these cases, we don't add a message mapping indicating the exception, but use the Status of SSP, because the different status codes of Status can be used to represent different exceptions, such as T28 and T29 are all pointing to P32. The different status codes of Status are all mapped to SIP 200 OK, which is made in order to simplify the model, otherwise the model will be very complex. In fact, when NOTIFY is sent to IMPS server through IWF, the NOTIFY has been transmitted successfully from Shared XDMS point of view. In resolving the conflicts in the marking M^1, besides using the methods used to resolve the conflicts in the other markings, we add P39 to distinguish the difference of the tokens arrived at P38 and P39, in order to restrict the happening of different transitions, i.e. restrict the happening of T28 and T29, which is also a good way to resolving the conflict.

We make the invariant analysis for the model shown in Figure 4 by PIPE2. The five nonnegative T-Invariants exported from PIPE2 are shown by matrix *J*, the two nonnegative S-Invariants exported from PIPE2 are shown by matrix *I*, as shown in Figure 5.

From the nonnegative T-Invariants shown in Figure 5, the Petri net model is covered by nonnegative T-Invariants, so it is bounded, live and resetable.

The equation of S-Invariants got from the two nonnegative S-Invariants are:

$$M(P_1) + M(P_2) + M(P_3) + M(P_4) + M(P_5)=1 \quad (1)$$

$$M(P_1) + M(P_2) + M(P_3) + M(P_5) + M(P_8) + M(P_9) + M(P_{10}) + M(P_{16})=1 \quad (2)$$

The equation (1) describes the states of the places in IWF-1 within Shared XDMS, which is consistent with the assumed execution result of the mapping for Shared XDMS and meets the requirement of protocol conversion for S-Invariants. The equation (2) describes the inter-working part in IWF-1 between Shared XDMS and IWF, which is also consistent with the assumption and indicates that in the process of the inter-working there has sequence for Shared XDMS, the external channel and IWF to process NOTIFY and unsubscribing request after NOTIFY has been sent out, i.e. it ensures that the received NOTIFY can be processed after the unsubscribing request has been received, so as to ensure the service security of Shared XDMS and IWF in the inter-working. Because the S-Invariants above are just the bases of the S-Invariants, the other S-Invariants can be constructed by the linear combinations of the above S-Invariants. We make the test for the other S-Invariants, and the result of test indicates that two place sets for processing SIP and SSP protocols in IWF are corresponding to two S-Invariants, four place sets for processing XCAP and SSP protocols in IWF are corresponding to four S-Invariants, the place sets for processing the communications between Shared XDMS and IWF are corresponding to two S-Invariants, the place set for processing the communications between IMPS server and IWF is corresponding to one S-Invariants.

As shown in the above analysis, the Petri net model meets all properties of a correct Petri net model, it is reasonable and viable for the mapping proposed for the XDM service. The execution of the coupled Petri net model can prove that the model can find out and resolve the possible exception, the added IWF-4 and IWF-5 can completely work well with the

existed reference points (IWF-1, IWF-2 and IWF-3), so it is reasonable and viable for the Enhanced Architectural Model proposed in (Zhang, 2007).

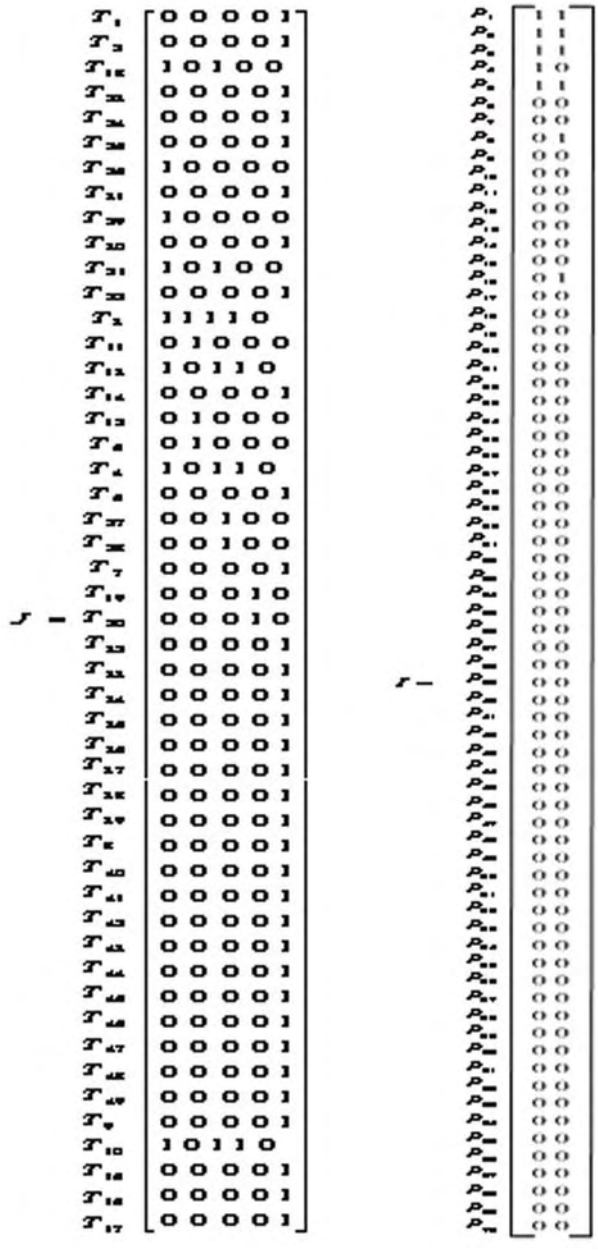


Fig. 5. The nonnegative T-Invariants and nonnegative S-Invariants in the coupled Petri net model

5. Resolving of conflict

In the process of the Petri net modeling, we don't use the methods proposed in (Zhu, 2007) for resolving the conflicts, but resolve the conflicts according to the real service execution. There are two concrete methods in this chapter: adding service mapping, adding place (i.e. adding the state of service execution) to restrict the happening of the transitions, such as P39. Besides the two methods, there are some other methods to resolve the conflicts, such as the methods proposed in (Lin, 2005): importing priority, giving different predications of implementation condition, giving different implementation time of the transition, giving different implementation possibility of a transition, and so on; the methods proposed in (Zhu, 2006a; Zhu, 2007): adding complementary place and side place, importing inhibitor arc and static testing arc, and so on.

6. Conclusions

In this chapter, with the procedure of Protocol Conversion Methodology, a Petri net model is constructed to verify the mapping and the Enhanced Architectural Model proposed in (Zhang, 2007), find and exclude the possible exceptions in the inter-working. After the strict mathematical analysis and verification for the model, which prove that the model meets all properties of a correct Petri net model, the mapping and the Enhanced Architectural Model are proved to be reasonable and viable, and the probable exceptions in the inter-working can be found and excluded. During the modeling experiences of the inter-working with Petri Nets, a new coupling criteria for Petri net and some new methods for solving the conflict of a Petri Net are proposed, and the methodology is summarized, which enriches the application of Petri Nets for the Protocol Conversion Methodology.
There are many standards or solutions for XDM, as the concept of XDM is almost same among different standards or solutions, the inter-working model proposed in this chapter has highly universal value and can provide an applicable reference for the inter-working between other standards, such as the inter-working between SIMPLE and XMPP.

7. Acknowledgement

This work was jointly supported by: (1) National Science Fund for Distinguished Young Scholars (No. 60525110); (2) National 973 Program (No. 2007CB307100, 2007CB307103); (3) Program for New Century Excellent Talents in University (No. NCET-04-0111); (4) Development Fund Project for Electronic and Information Industry (Mobile Service and Application System Based on 3G); (5) National Specific Project for Hi-tech Industrialization and Information Equipments (Mobile Intelligent Network Supporting Value-added Data Services).

8. References

3GPP.(2002). TS 22.250, IP Multimedia Subsystem (IMS) group management, Stage 1(Release 6)
3GPP.(2004). TS 24.841, Presence service based on Session Initiation Protocol (SIP); Functional models, information flows and protocol details, Stage 3 (Release 6)
3GPP.(2005a). TS 22.340, IP Multimedia System (IMS) messaging, Stage 1 (Release 7)
3GPP.(2005b). TS 22.940, IP Multimedia System (IMS) messaging, Stage 1 (Release 7)
3GPP.(2005c). TS 24.247, Messaging service using the IP Multimedia (IM) Core Network

(CN) subsystem, Stage 3 (Release 6)
3GPP.(2005d). TS 22.141, Presence Service, Stage 1(Release 7)
3GPP.(2005e). TS 23.141, Presence Service; Architecture and functional description, Stage 1(Release 7)
3GPP.(2005f). TS 24.141, Presence service using the IP Multimedia (IM) Core Network (CN) subsystem, Stage 3 (Release 7)
3GPP2.(2002). S.R0062-0, Presence for Wireless Systems Stage 1 Requirements, V1.0, 2002
3GPP2.(2004). X.S0027-001-0, Presence Service; Architecture and functional description, V1.0, 2004
3GPP2.(2005a). X.P0027-004-0, Network Presence, V1.0, 2005
3GPP2.(2005b). X.S0027-003-0, Presence Service using IP Multimedia Core Network Subsystem; Stage 3, V1.0, 2005
3GPP2.(2006). X.P0013-016, Messaging service using the IP Multimedia (IM) Subsystem (IMS); V0.7, 2006
Day, M. ; Rosenberg, J. ; & H. Sugano.(2000a). A Model for Presence and Instant Messaging, RFC 2778, February 2000
Day, M. ; Aggarwal, S. ; Mohr, G. & Vincent J.(2000b). Instant Messaging / Presence Protocol Requirements, RFC 2779, February 2000
Green Jr P E.(1988). Protocol conversion. *Network Interconnection and Protocol Conversion*, IEEE Press, 1988, pp2-13. New York
Lin Chang.(2005). *Stochastic Petri Net and System Performance Evaluation.(the Second Edition)*, Tsinghua University Press, ISBN 7-302-10651-7, Beijing
Open Mobile Alliance.(2005a). IMPS-SIP/SIMPLE Interworking Function Architecture, Draft Version 0.2, 2005-05-20
Open Mobile Alliance(2005b). IMPS SIP/SIMPLE Interworking Function Requirements, Draft Version 1.0, 30 August 2005
Open Mobile Alliance.(2006a). XML Document Management Requirements, V1.0, 12 Jun 2006
Open Mobile Alliance.(2006b). XML Document Management Architecture, V1.0, 12 Jun 2006
Open Mobile Alliance.(2006c). Enabler Release Definition for XML Document Management, V1.0.1, 28 Nov 2006
Open Mobile Alliance.(2006d). XML Document Management (XDM) Specification, V1.0.1, 28 Nov 2006
Open Mobile Alliance.(2006e). Shared XDM Specification, V1.0.1, 28 Nov 2006
Open Mobile Alliance.(2006f). Shared Group XDM Specification, Draft V 2.0, 18 Dec 2006
Open Mobile Alliance.(2006g). Shared List XDM Specification, Draft V 2.0, 18 Dec 2006
Open Mobile Alliance.(2006h). OMA XML Document Management Requirements, Draft V 2.0, 19 Dec 2006
Open Mobile Alliance.(2006i). XML Document Management Architecture, Draft V 2.0, 19 Dec 2006
Open Mobile Alliance.(2006j). XML Document Management (XDM) Specification, Draft V 2.0, 19 Dec 2006
Open Mobile Alliance.(2006k). Shared Profile XDM Specification, Draft V 2.0, 19 Dec 2006
Open Mobile Alliance.(2006l). Enabler Release Definition for XML Document Management, Draft V 2.0, 20 Dec 2006
Open Mobile Alliance.(2007a). IMPS Architecture, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007b). OMA IMPS Delta Requirements, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007c). Presence Attributes, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007d). Client-Server Protocol Session and Transactions, V 1.3, 23 Jan

2007
Open Mobile Alliance.(2007e). Server-Server Protocol Semantics, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007f). Enabler Release Definition for IMPS, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007g). Client-Server Protocol Data Types, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007h). Client-Server Protocol Plain Text Syntax, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007i). Client-Server Protocol Transport Bindings, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007j). Client-Server Protocol XML Syntax, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007k). Presence Attributes XML Syntax, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007l). Server-Server Protocol Transport Binding, V 1.3, 23 Jan 2007
Open Mobile Alliance.(2007m). Server-Server Protocol XML Syntax, V 1.3, 23 Jan 2007
P. Saint-Andre, Ed.(2004a). Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920, October 2004
P. Saint-Andre, Ed.(2004b). Extensible Messaging and Presence Protocol (XMPP):Instant Messaging and Presence. RFC 3921, October 2004
Rosenberg, J. ; Schulzrinne, H. ; Camarillo, H. ; Johnston, A. ; Peterson,J. ; Sparks, R. ; Handley, M. & E. Schooler.(2002). SIP: Session Initiation Protocol, RFC 3261, June 2002
Rosenberg, J.(2005). Extensible Markup Language (XML) Formats for Representing Resource Lists, draft-ietf-simple-xcap-list-usage-05, 2005.2
Rosenberg, J.(2006). The Extensible Markup Language (XML) Configuration Access Protocol, draft-ietf-simple-xcap-12(work in progress), October 2006
Zhang Yuting ; Liao Jianxin ; Zhu Xiaomin ; Wu Wei & Ma Jun.(2007). Inter-working between SIMPLE and IMPS. *Computer Standards & Interfaces*, Vol. 29, No. 5, (July 2007) page numbers (584-600), ISSN:0920-5489
Zhu Xiaomin; Liao Jianxin ; Wang Peng & Wang Jianbin.(2006a). Modelling Click-to-Dial Service with Petri Nets. *Journal of Electronics & Information Technology*, Vol. 28, No. 3, (March 2006) page number (552-556), ISSN:1009-5896
Zhu Xiaomin; Liao Jianxin & Chen Junliang.(2006b). Improved Protocol Conversion Methodology and Its Application. *International Journal of Computers and Applications*, Vol. 28, No. 3, (September 2006) page numbers(210-221), ISSN:1206-212X
Zhu Xiaomin; Liao Jianxin & Chen Junliang.(2007). Petri Net Model of Protocol Conversion for CTF service: Its Universal Coupling Criteria and Property Analysis. *International Journal of Communication Systems*, Vol. 20, No. 5, (May 2007) page numbers (533-551), ISSN:1074-5351

9. Links

1. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) IETF Working Group
   http://www.ietf.org/html.charters/simple-charter.html
2. WV white paper
   http://www.openmobilealliance.org/tech/affiliates/wv/wvindex.html
3. Petri Nets World
   http://www.informatik.uni-hamburg.de/TGI/PetriNets/, March 2006
4. Visual Object Net++ Homepage[DB/OL]
   http://www.systemtechnik.tu-ilmenau.de/~drath/visual_E.htm, Jun. 2003
5. Platform Independent Petri Net Editor 2 Homepage
   http://pipe2.sourceforge.net/, May 2003

5

Modelling Systems by Hybrid Petri Nets: an Application to Supply Chains

Mariagrazia Dotoli[1], Maria Pia Fanti[1], Alessandro Giua[2] and Carla Seatzu[2]
[1]*Dip. di Elettrotecnica ed Elettronica, Politecnico di Bari,*
[2]*Dip. di Ingegneria Elettrica ed Elettronica, Università degli Studi di Cagliari*
Italy

1. Introduction

Petri Nets (PNs) are a discrete event model firstly proposed by C. A. Petri in his Ph.D. thesis in the early 1960s (Petri, 1962). The main feature of a (discrete) PN is that its state is a vector of non-negative integers. This is a major advantage with respect to other formalisms such as automata, where the state space is a symbolic unstructured set, and has been exploited to develop many analysis techniques that do not require to enumerate the state space (structural analysis) (Silva et al., 1996). Another key feature of PNs is their capacity to graphically represent and visualize primitives such as parallelism, concurrency, synchronization, mutual exclusion, etc.
In the related literature various PN extensions have been proposed. In this paper we focus on Continuous and Hybrid PNs.
Continuous Petri Nets (CPNs) originate from the "fluidification" of discrete PNs (David & Alla, 1987). In simple words, the content of places is relaxed to be a real non-negative number rather than an integer non-negative number, and appropriate rules for transitions firings are given. This highly reduces the computational complexity of the analysis and optimization of realistic scale problems, and has been successfully applied to manufacturing systems. The main advantages of fluidification can be summarized in the following four items.

- The computational complexity of the analysis and control of complex systems may be significantly reduced.
- Fluid approximations provide an aggregate formulation to deal with complex systems, thus reducing the dimension of the state space. The resulting simple structures allow explicit computation and performance optimization.
- The design parameters in fluid models are continuous; hence, it is possible to use gradient information to speed up optimization and to perform sensitivity analysis.
- Finally, in many cases it has also been shown that fluid approximations do not introduce significant errors when carrying out performance analysis via simulation.

In general, different fluid approximations are necessary to describe the same system, depending on its discrete state, e.g., in the manufacturing domain, machines working or down, buffers full or empty, and so on. Thus, the resulting model can be better described as a hybrid model, where a different continuous dynamics is associated to each discrete state.
Hybrid Petri Nets (HPNs) keep all those good features that make discrete PNs a valuable

discrete-event model: they do not require the exhaustive enumeration of the state space and can finitely describe systems with an infinite state space; they allow modular representation where the structure of each module is kept in the composed model; the discrete state is represented by a vector and not by a symbolic label, thus linear algebraic techniques may be used for their analysis. Different HPN models have been proposed in the literature, but there is so far no widely accepted classification of such models.
In Section 2 we provide a brief survey of the most important HPN models presented in the related literature. The main theoretical results and the main application areas within each framework are also mentioned. We recently provided a more detailed survey in (Dotoli et al., 2007).
In Section 3 we focus our attention on a particular model of HPNs, called *First-Order Hybrid Petri Nets* (FOHPNs) because its continuous dynamics are piece-wise constant. FOHPNs were originally proposed in (Balduzzi et al., 2000) and have been efficiently used in many application domains, such as manufacturing systems (Balduzzi et al., 2001; Giua et al., 2005) and inventory control (Furcas et al., 2001). Interesting optimization problems have also been studied considering real applications, such as a bottling plant (Giua et al., 2005) and a cheese factory (Furcas et al. 2001).
Finally, in Section 4 we show how FOHPNs can be efficiently used for modelling and controlling large and complex systems such as *Supply Chains* (SCs). SCs are complex emerging distributed manufacturing systems whose analysis, design and management is currently an active area of research (Viswanadham & Gaonkar, 2003; Viswanadham & Raghavan, 2000; Dotoli et al., 2005; Dotoli et al., 2006). More precisely, a SC is defined as a collection of independent companies possessing complementary skills and integrated with transportation and storage systems, information and financial flows, with all entities collaborating to meet the market demand. Appropriate modelling and analysis of such highly complex systems are crucial for performance evaluation and to compare competing SCs. However, in the related literature few contributions deal with the problem of modelling and analyzing the SC operational behaviour. Viswanadham and Raghavan (2000) model SCs as discrete event dynamical systems, in which the evolution depends on the interaction of discrete events such as the arrival of the components at the facilities, the departure of the transport, the start of the operations at the manufacturers and the assemblers. In (Desrochers et al., 2005) a two-product SC is modelled by complex-valued token PNs and the performance measures are determined by simulation. However, the limit of such formalisms is the modelling of products or batches of parts by means of discrete quantities (i.e., tokens). This assumption is not realistic in large SCs with a huge amount of material flow. Hence, this paper uses FOHPNs to model and manage SCs. Using a modular approach based on the idea of bottom-up methodology (Zhou & Venkatesh, 1998), this work develops a modular FOHPN model of SCs where the input buffers are managed by the well known fixed order quantity policy. In particular, transporters and manufacturers are described by continuous transitions, buffers are continuous places, and products are represented by continuous flows (fluids) routing from manufacturers, buffers and transporters.

2. Hybrid Petri nets

The first fluid PN model is the so called "Continuous and Hybrid Petri Net" model introduced by R. David and H. Alla in their seminal paper (David & Alla, 1987). Based on

this first formalism, and motivated by particular applications, a family of extended hybrid models has then been proposed in the literature. In this section we briefly recall some of them, namely Fluid Stochastic Petri Nets, Batch Nets, DAE-Petri Nets, Hybrid Flow Nets, Differential Petri Nets and High-Level Hybrid Nets. For a more detailed survey on Hybrid Petri Nets (HPNs) we address the reader to (Dotoli et al., 2007) and to (David & Alla, 2005).

2.1 Continuous and hybrid Petri nets

All the works collected under this heading are based on or directly inspired to the model presented by R. David and H. Alla in the late eighties (David & Alla, 1987). These authors have obtained a *continuous model* by *fluidification*, i.e., by relaxing the condition that the marking be an integer vector. *Hybrid Petri nets* are then made of a "continuous part" (continuous places and transitions) and a "discrete part" (discrete places and transitions). The continuous part can model systems with continuous flows and the discrete part models the logic behavior.

Several contributions in this framework have been presented in the last decade, as well as some interesting extensions with respect to the original model.

As an example, the problem of determining an optimal stationary mode of operation for a system described by a timed CPN has been studied in (Gaujal & Giua, 2004). Some characterizations of equilibrium points in steady-state are given in (Mahulea et al., 2007), where an optimal steady-state control is also studied. An interesting comparison on two different techniques to compute the steady-state of continuous nets was made in (Demongodin & Giua, 2002): a method based on linear programming and a method based on graph theory are considered.

Other interesting papers have been devoted to the problem of production frequencies estimation for systems that are modeled by CPNs (Lefebvre, 2000), to the design of observers (Júlvez et al., 2004), to the reachability analysis (Júlvez et al., 2003), to the stability analysis (Amer-Yahia & Zerhouni, 2001), and to the deadlock-freeness analysis (Júlvez et al., 2002).

The problem of deriving an optimal control law for CPNs under the assumption of *finite servers semantics* has been studied in (Bemporad et al., 2004). In (Mahulea et al., 2006a) the authors considered timed CPNs under *infinite servers semantics* that usually provide a much better approximation of the discrete system than finite servers semantics (Mahulea et al., 2006b). They deal with the problem of controlling CPNs in order to reach a final (steady state) configuration while minimizing a quadratic performance index.

CPNs have been mainly applied in the manufacturing domain (for an exhaustive list of references see (Dotoli et al., 2007)), even if some other interesting applications have been presented, like (Amer-Yahia et al., 1997) dealing with biological systems, and (Júlvez & Boel, 2005) dealing with transportation systems.

FOHPNs follow the formalism described in (Alla & David, 1998) with the addition of algebraic analysis techniques, and have been firstly presented in (Balduzzi et al., 2000). FOHPNs consist of continuous places holding fluid, discrete places containing a non-negative integer number of tokens, and transitions, either discrete or continuous. As in all hybrid models, in FOHPNs the authors distinguish two behavioral levels: time-driven and event-driven. The continuous time-driven evolution of the net is described by first-order fluid models, i.e., models in which the continuous flows have constant rates and the fluid content of each continuous place varies linearly with time. A discrete-event model describes

the behaviour of the net that, upon the occurrence of macro-events, evolves through a sequence of macro-states. The authors set up a linear algebraic formalism to study the first-order continuous behavior of this model and show how its control can be framed as a conflict resolution policy that aims at optimizing a given objective function. The use of linear algebra leads to sensitivity analysis that allows one to study how changes in the structure of the model influence the optimal behavior. This model is extensively presented in the rest of this paper.

2.2 Other models

The *Fluid Stochastic Petri Net* (FSPN) model has been firstly presented by K.S. Trivedi and V.G. Kulkarni in the early nineties (Trivedi & Kulkarni, 1993). Here the authors extend the stochastic Petri nets framework (Ajmone Marsan et al., 1995) to FSPNs by introducing places with continuous tokens and arcs with fluid flow so as to handle stochastic fluid flow systems. No continuous transitions are present in this model, and the set of transitions is partitioned into timed transitions and immediate transitions, where timed transitions have an exponentially distributed firing time. They define hybrid nets in such a way that the discrete and continuous portions may affect each other.

Batch Petri Nets (BPNs) represent a formalism derived in (Demongodin et al., 1998) as a modeling tool for the particular class of *batch processes*. It intends to model variable delays on continuous flows by adding to a hybrid Petri net special nodes called *batch nodes*. Batch nodes combine both a discrete event and a linear continuous dynamic behaviour in a single structure. Evolution rules are determined in order to carry out the simulation of systems based on accumulation phenomena, thus the resulting formalism is well suited to model high throughput production lines.

Differential Algebraic Equations-Petri Nets (DAE-PNs) are based on the model presented in (Andreu et al., 1996; Champagnat et al., 1998; Valentin-Roubinet, 1998). This approach does not try to represent in a unified way the continuous and discrete aspects, as it is the case in HPNs. On the contrary, the model focuses on the interaction between a discrete Petri net model that captures the discrete behaviour of a batch system, and a continuous model, which is a set of differential algebraic equations. DAE-PNs can be seen as an extension of hybrid automata (Alur et al., 1993; Puri & Varaiya, 1996). This approach is well suited for modelling batch processes where it is necessary to concurrently deal with continuous and discrete models. It has also been tested in the food industry for the validation of scheduling policies and has been developed for supervisory control and reactive scheduling.

Hybrid Flow Nets (HFNs) have been proposed in (Flaus, 1997; Flaus & Alla, 1997). This approach is based on the analysis of a system as a set of continuous and discrete flows. The notion of HFNs can then be seen as an extension of PNs for hybrid systems. This modeling tool is made of a continuous flow net interacting with a PN according to a control interaction. The overall philosophy of PNs is preserved again. The discrete part is a PN while the continuous part is called *continuous flow* net, whose dynamic evolution has to be defined so as to be similar to the one of PNs, with a continuous enabling rule and a continuous firing rule. HFNs are well suited for the modeling and control of industrial transformation processes, for which the dynamics behavior has a hybrid nature.

Differential Petri Nets (DPNs) have been firstly presented in (Demongodin & Koussoulas, 1998). The main feature of this class of PNs is that it allows us to model continuous-time dynamic processes represented by a finite number of linear first-order differential state

equations. The DPN is defined through the introduction of a new kind of place and transition, namely, the *differential place* and the *differential transition*. The marking of the differential place represents a state variable of the continuous system that is modeled. A firing speed, representing either a variable proportional to a state variable or an independent variable, is associated to every differential transition. A differential transition is always enabled, thus to discretize the continuous system; a firing frequency, representing the integration step that would be used when carrying out an integration of the differential equation, is associated to any differential transition. Evolution rules have been developed to specify the simulation of hybrid systems composed by a continuous part cooperating with a discrete event part, i.e., the typical paradigm of a supervisory control system.
Finally, under the heading *High-Level Hybrid Petri Nets* (HLHPNs) we collect different models presented by several authors (Chen & Hanisch, 1998; Genrich & Schuart, 1998; Giua & Usai, 1998). All these models, however, are based on high-level nets, i.e., nets characterized by the use of *structured individual tokens*. HLHPNs are a useful model that provides a simple graphical representation of hybrid systems and takes advantage of the modular structure of PNs in giving a compact description of systems composed of interacting subsystems, both time-continuous and discrete-event. The use of colors in the continuous places allows one to model continuous variables that may take negative values.

3. First-order hybrid Petri nets

In this section we provide a detailed presentation of the FOHPN model (Balduzzi et al., 2000). For a more comprehensive introduction to place/transition PNs see (Murata, 1989).

3.1 Net structure

A FOHPN is a structure

$$N = (P,T,Pre,Post, D, C).$$

The set of *places* $P = P_d \cup P_c$ is partitioned into a set of *discrete* places P_d (represented as circles) and a set of *continuous* places P_c (represented as double circles). The cardinality of P, P_d and P_c is denoted n, n_d and n_c, respectively. We assume that the place labeling is such that: $P_c = \{ p_i \mid i=1, \dots , n_c \}$, $P_d = \{ p_i \mid i= n_c+1, \dots , n\}$.
The set of *transitions* $T = T_d \cup T_c$ is partitioned into a set of discrete transitions T_d and a set of continuous transitions T_c (represented as double boxes). The set $T_d = T_I \cup T_D \cup T_E$ is further partitioned into a set of *immediate* transitions T_I (represented as bars), a set of *deterministic timed* transitions T_D (represented as black boxes), and a set of *exponentially distributed timed* transitions T_E (represented as white boxes). The cardinality of T, T_d and T_c is denoted q, q_d and q_c, respectively. We also denote with q_t the cardinality of the set of timed transitions $T_t = T_D \cup T_E$. We assume that the transition labeling is such that: $T_c = \{t_j \mid j=1, \dots , q_c \}$, $T_t = \{t_j \mid j= q_c+1, \dots , q_c+q_t\}$, $T_I = \{t_j \mid j= q_c+q_t+1, \dots , q\}$.
The *pre-* and *post-incidence functions* that specify the arcs are (here $R_0^+ = R^+ \cup \{0\}$):

$$Pre, Post : \begin{cases} P_c \times T \to R_0^+ \\ P_d \times T \to N \end{cases}$$

We require (*well-formed nets*) that for all $t \in T_c$ and for all $p \in P_d$, $Pre(p,t) = Post(p,t)$. This ensures that the firing of continuous transitions does not change the marking of discrete places.

The function $D : T_t \rightarrow R^+$ specifies the timing associated to timed discrete transitions. We associate to a deterministic timed transition $t_j \in T_D$ its (constant) firing delay $\delta_j = D(t_j)$. We associate to an exponentially distributed timed transition $t_j \in T_E$ its average firing rate $\lambda_j = D(t_j)$: the random delay is distributed according to the probability density function $f_j(\tau) = \lambda_j \exp(-\lambda_j \tau)$ and the average firing delay is $1/\lambda_j$.

The function $C : T_c \rightarrow R_0^+ \times R_\infty^+$ specifies the *firing speeds* associated to continuous transitions (here $R_\infty^+ = R^+ \cup \{+\infty\}$). For any continuous transition $t_j \in T_c$ we let $C(t_j) = (V_j', V_j)$, with $V_j' \leq V_j$. Here V_j' represents the *minimum firing speed* (mfs) and V_j represents the *Maximum Firing Speed* (MFS). In the following, unless explicitly specified, the mfs of a continuous transition t_j will be $V_j'=0$.

We denote the preset (postset) of transition t as ${}^\bullet t$ ($t^\bullet$) and its restriction to continuous or discrete places as ${}^{(d)}t = {}^\bullet t \cap P_d$ or ${}^{(c)}t = {}^\bullet t \cap P_c$. A similar notation may be used for presets and postsets of places. The *incidence matrix* of the net is defined as $C(p,t) = Post(p,t) - Pre(p,t)$. The restriction of C to P_X and T_Y ($X,Y \in \{c,d\}$) is denoted C_{XY}. Note that by the well-formedness hypothesis $C_{dc} = 0_{nd \times qc}$.

3.2 Marking and enabling

A *marking*

$$m : \begin{cases} P_c \rightarrow R_0^+ \\ P_d \rightarrow N \end{cases}$$

is a function that assigns to each discrete place a non-negative integer number of tokens, represented by black dots, and assigns to each continuous place a fluid volume; m_i denotes the marking of place p_i. The value of the marking at time τ is denoted $m(\tau)$. The restrictions of m to P_d and P_c are denoted with m^d and m^c, respectively.

An *FOHPN system* $\langle N, m(\tau_0) \rangle$ is an FOHPN N with an initial marking $m(\tau_0)$.

The enabling of a discrete transition depends on the marking of all its input places, both discrete and continuous.

Definition 3.1 Let $\langle N, m \rangle$ be an FOHPN system. A discrete transition t is *enabled* at m if for all $p_i \in {}^\bullet t$, $m_i \geq Pre(p_i,t)$.

A continuous transition is enabled only by the marking of its input discrete places. The marking of its input continuous places, however, is used to distinguish between strongly and weakly enabling.

Definition 3.2 Let $\langle N, m \rangle$ be an FOHPN system. A continuous transition t is *enabled* at m if for all $p_i \in {}^{(d)}t$, $m_i \geq Pre(p_i,t)$.

We say that an enabled transition $t \in T_c$ is:

- *strongly enabled* at m if for all places $p_i \in {}^{(c)}t$, $m_i > 0$;
- *weakly enabled* at m if for some $p_i \in {}^{(c)}t$, $m_i = 0$.

3.3 Net dynamics

We now describe the dynamics of an FOHPN. First, we consider the behaviour associated to discrete transitions that combines a continuous dynamics associated to the timers, and a

discrete-event dynamics associated to the transition firing. Then we consider the time-driven behaviour associated to the firing of continuous transitions.
Note that the evolution of an FOHPN is characterized by the occurrence of some events that we call *macro-events*, while the time interval between two consecutive macro-events is called a *macro-period*. As discussed in detail in the following two paragraphs, macro-events may be either related to the firing and/or the enabling condition of discrete transitions, or to the enabling condition and/or the enabling state of a continuous transition.
In the following we use $e_{i,r}$ to denote the ith canonical basis vector of dimension r. We also define, to simplify the notation, the index $\rho(j)=j-q_c$ that is used to define the firing vector associated to a discrete transition.

3.3.1 Discrete transitions dynamics

We associate to each timed transition $t_j \in T_t$ a timer υ_j.
Definition 3.3 [Timers evolution] Let $\langle N, m\rangle$ be an FOHPN system and $[\tau_k,\tau)$ be an interval of time in which the enabling state of a transition $t_j \in T_t$ does not change. If t_j is enabled in this interval then

$$\upsilon_j(\tau)= \upsilon_j(\tau_k)+ (\tau- \tau_k)$$

while if t_j is not enabled in this interval then

$$\upsilon_j(\tau)= \upsilon_j(\tau_k)=0.$$

Whenever t_j is disabled or it fires, its timer is reset to 0.
With the notation of (Ajmone Marsan et al., 1995), we are using a *single-server* semantics, i.e. only one timer is associated to each timed transition, and an *enabling-memory* policy, i.e. each timer is reset to 0 whenever its transition is disabled. The approach we present, however, can also be easily extended to take into account infinite server semantics.
The vector of timers associated to timed transitions is denoted

$$\upsilon=[\, \upsilon_{qc+1}, \upsilon_{qc+2}, \ldots, \upsilon_{qc+qt}\,]^T \in (R_0^+)^{qt}.$$

Note that the timer evolution is continuous and linear during a *macro – period* and may change at the occurrence of the following *macro-events*:

1. a discrete transition fires, thus changing the discrete marking and enabling (or disabling) a timed transition;
2. a continuous place reaches a fluid level that enables (or disables) a discrete transition.

An enabled timed transition $t_j \in T_t$ fires when the value of its timer reaches a given value $\upsilon_j(\tau)= \upsilon_j^*$: we call υ_j^*'s the *timer set points*. In the case of a deterministic transition, $\upsilon_j^* =\delta_j$ is the associated delay. In the case of a stochastic transition, υ_j^* is the current sample of the associated random variable: it is drawn each time the transition is newly enabled. An immediate transition fires as soon as it is enabled, i.e. it can be considered as a deterministic transition with $\upsilon_j^* =0$.
Definition 3.4 [Discrete transition firing] The firing of a discrete transition t_j at $m(\tau^-)$ yields the marking $m(\tau)$ and for each place p it holds $m_p(\tau) = m_p(\tau^-) + Post(p, t_j) - Pre(p, t_j) = m_p(\tau^-)+ C(p, t_j)$. Thus we can write

$$\begin{cases} m^c(\tau) = m^c(\tau^-) + C_{cd}\sigma(\tau) \\ m^d(\tau) = m^d(\tau^-) + C_{dd}\sigma(\tau) \end{cases}$$

where $\sigma(\tau) = e_{\rho(j),qd}$ is the *firing count vector* associated to the firing of transition t_j.
In the above definition we note that a transition t_j is the $\rho(j)$th discrete transition, hence, say, $C_{cd}\, e_{\rho(j),qd}$ represents the column of matrix C_{cd} corresponding to transition t_j.

3.3.2 Continuous transitions dynamics

The *Instantaneous Firing Speed* (IFS) at time τ of a transition $t_j \in T_c$ is denoted $v_j(\tau)$. We can write the equation which governs the evolution in time of the marking of a place $p_i \in P_c$ as

$$\dot{m}_i(\tau) = \sum_{t_j \in T_c} C(p_i, t_j) v_j(\tau) = e_{i,n_c}{}^T C_{cc} v(\tau) \tag{1}$$

where $v(\tau) = [v_1(\tau), \ldots, v_{nc}(\tau)]^T$ is the IFS vector at time τ. Indeed, equation (1) holds assuming that at time τ no discrete transition is fired and that all speeds $v_j(\tau)$ are continuous in τ.
The enabling state of a continuous transition t_j defines its admissible IFS v_j. In particular, three cases are alternatively possible.

- If t_j is not enabled then $v_j = 0$.
- If t_j is strongly enabled, then it may fire with any firing speed $v_j \in [V'_j, V_j]$.
- If t_j is weakly enabled, then it may fire with any firing speed $v_j \in [V'_j, V''_j]$, where the upper bound V''_j on the firing speed is such that $V''_j \leq V_j$ and depends on the flow entering the set of input continuous places $^{(c)}t_j$ that are empty. In fact, t_j cannot remove more fluid from any empty input continuous place p than the quantity entered in p by other transitions.

The computation of the IFS of enabled transitions is not a trivial task. We set up in the following Subsection 3.4 a linear—algebraic formalism to do this. Here we simply discuss the net evolution, assuming that the IFS are given.
We say that a *macro-event* occurs when either cases hold:

1. a discrete transition fires, thus changing the discrete marking and enabling (or disabling) a continuous transition;
2. a continuous place becomes empty, thus changing the enabling state of a continuous transition from strong to weak.

Definition 3.5 [Continuous transition firing] Let τ_k and τ_{k+1} be the occurrence times of two consecutive macro-events as defined above; we assume that within the interval of time $[\tau_k, \tau_{k+1})$ the IFS vector is constant and denoted $v(\tau_k)$. The continuous behaviour of an FOHPN for $\tau \in [\tau_k, \tau_{k+1})$ is described by

$$\begin{cases} m^c(\tau) = m^c(\tau_k) + C_{cc} v(\tau_k)(\tau - \tau_k) \\ m^d(\tau) = m^d(\tau_k) \end{cases}$$

3.4 Admissible IFS vectors

We use linear inequalities to characterize the set of *all* admissible firing speed vectors $\mathcal{S}$. Each IFS vector $v \in \mathcal{S}$ represents a particular mode of operation of the system described by the net. As discussed in detail in the subsequent Subsection 3.5, the system operator may choose, among all possible modes of operation, the best one according to a given objective.

The set of admissible IFS vectors form a convex set described by linear equations.

Definition 3.6 [Admissible IFS vector] Let $\langle N, m\rangle$ be an FOHPN system with n_c continuous transitions and incidence matrix C. Let

- $T_E(m) \subset T_c$ ($T_N(m) \subset T_c$) be the subset of continuous transitions enabled (not enabled) at m,
- $P_E(m) = \{ p \in P_c \mid m_p = 0 \}$ be the subset of empty continuous places.

Any *admissible IFS vector* $v = [v_1, ..., v_{nc}]^T$ at m is a feasible solution of the following linear set:

$$\begin{cases} (a) & V_j - v_j \geq 0 & \forall t_j \in T_E(m) \\ (b) & v_j - V'_j \geq 0 & \forall t_j \in T_E(m) \\ (c) & v_j = 0 & \forall t_j \in T_N(m) \\ (d) & \sum_{t_j \in T_E} C(p,t_j) \cdot v_j \geq 0 & \forall p \in P_E(m) \\ (e) & v_j \geq 0 & \forall t_j \in T_c \end{cases} \tag{2}$$

Apart from the non-negativity constraint *(e)*, the total number of constraints that define this set is: $2\ \text{card}\{T_E(m)\} + \text{card}\{T_N(m)\} + \text{card}\{P_E(m)\}$. The set of all feasible solutions is denoted $S(N,m)$.

Note that constraints of the form (2.a), (2.b) and (2.c) follow from the firing rules of continuous transitions. Constraints of the form (2.d) follow from (1), because if a continuous place is empty then its fluid content cannot decrease. Note that if $V'_i=0$, then the constraint of the form (2.b) associated to t_i reduces to a non-negativity constraint on v_i.

3.5 Control

In the previous section we have shown how appropriate linear inequalities can be used to define the set of all admissible firing speed vectors S. Each vector $v \in S$ represents a particular mode of operation of the system described by the net, and among all possible modes of operation, the system operator may choose the best one according to a given objective. Some examples are given in the following.

- **Maximize flows**. In an FOHPN we may consider as optimal the solution v^* of (2) that maximizes the performance index $J = 1^T \cdot v$, which is of course intended to maximize the sum of all flow rates. In the manufacturing domain this may correspond to maximizing machines utilization.
- **Maximize outflows**. In an FOHPN we may want to maximize the performance index $J = c^T \cdot v$, where

$$c_j = \begin{cases} 1 & \text{if } t_j \text{ is an exogenous transition} \\ 0 & \text{if } t_j \text{ is an endogeneous transition} \end{cases}$$

  In the manufacturing domain this may correspond to maximizing throughput.
- **Minimize stored fluid**. In an FOHPN we may want to minimize the derivative of the marking of a place $p \in P_c$. This can be done by minimizing the performance index $J = c^T \cdot v$, where

$$c_j = \begin{cases} C(p,t_j) & \text{if } t_j \in p^{(c)} \cup^{(c)} p \\ 0 & \text{otherwise} \end{cases}$$

In the manufacturing domain this may correspond to minimizing the work-in-process.
Note that this approach has several advantages with respect to other approaches proposed in the literature, e.g., (Dubois et al., 1994), where an iterative algorithm is given to determine one admissible vector. In fact, we can explicitly define the set of all admissible IFS vectors in a given macro-state and not just compute a particular vector. Then, we compute a particular (optimal) IFS vector solving a Linear Programming Problem (LPP), rather than by means of an iterative algorithm, whose convergence properties may not be good.
However, the above control procedure still suffers from a serious drawback. In fact, the set S corresponds to a particular system macro-state. Thus, our optimization scheme can only be *myopic*, in the sense that it generates a piece-wise optimal solution, i.e. a solution that is optimal only in a macro-period.
At present, we are looking for alternative solutions that are not myopic, but this is still an open issue. We believe that the approach used in (Bemporad et al., 2004) to optimally control CPNs could be be successfully applied also in the case of FOHPNs, but we still have to verify this conjecture.

4. Modelling and simulation of supply chains

This section shows the efficiency of FOHPNs in modelling and controlling at the operational level large and complex systems such as SCs.

4.1 The SC system description

The SC structure is typically described by a set of facilities with materials that flow from the sources of raw materials to manufacturers and onwards to assemblers and consumers of finished products. SC facilities are connected by transporters of materials, semi-finished goods and finished products. More precisely, the SC entities can be summarized as follows.

1. *Suppliers:* a supplier is a facility that provides raw materials, components and semi-finished products to manufacturers that make use of them.
2. *Manufacturers and assemblers*: manufacturers and assemblers are facilities that transform input raw materials/components into desired output products.
3. *Logistics and transporters*: storage systems and transporters play a critical role in distributed manufacturing. The attributes of logistics facilities are storage and handling capacities, transportation times, operation and inventory costs.
4. *Distributors*: distributors are intermediate nodes of material flows representing agents with exclusive or shared rights for the marketing of an item.
5. *Retailers or customers*: retailers or customers are sink nodes of material flows.

Here, part of the logistics, such as storage buffers, is considered pertaining to manufacturers, suppliers and customers. Moreover, transporters connect the different stages of the production process.
The dynamics of the distributed production system is traced by the flow of products between facilities and transporters. Because of the large amount of material flowing in the system, we model a SC as a hybrid system: the continuous dynamics models the flow of products in the SC, the manufacturing and the assembling of different products and its

storage in appropriate buffers. Hence, the levels of buffers accommodating products are represented by continuous states describing the amount of fluid material that the resources store. Moreover, we consider also discrete events occurring stochastically in the system, such as:

1. the blocking of the raw material supply, e.g. modeling the occurrence of labor strikes, accidents or stops due to the shifts;
2. the blocking of transport operations due to the shifts or to unpredictable events such as jamming of transportation routes, accidents, strikes of transporters, etc.;
3. the beginning and the end of a request from a SC facility.

4.2 A modular SC model based on FOHPNs

This section recalls a modular approach using FOHPNs to model SCs based on the idea of the bottom-up approach (Zhou & Venkatesh, 1998). Such a method can be summarized in two main steps: decomposition and composition. Decomposition involves dividing a system into several subsystems. As shown in (Dotoli et al., 2007), in SCs this division can be performed based on the determination of distributed system facilities (i.e., suppliers, manufacturers, assemblers, transporters, distributors, buffers and customers). All these subsystems are modelled by FOHPNs. Finally, composition involves the interacting of these sub-models into a complete model, representing the whole SC.

In the following we present the FOHPN models of the elementary subsystems composing a generic SC.

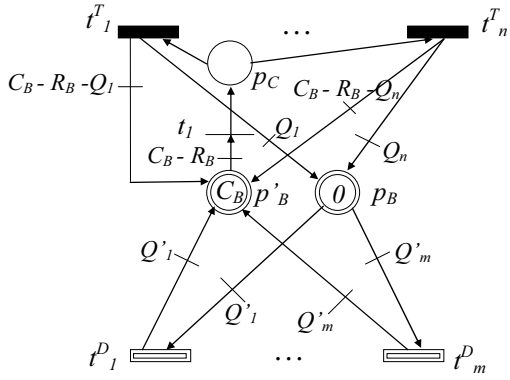
4.2.1 The inventory management model

Inventory management addresses two fundamental issues: when a stock should replenish its inventory and how much it should order from suppliers for each replenishment (Chen et al., 2005). Inventory systems with independent demand can use Fixed Order Quantity (FOQ) policies that manage inventory by placing an order of fixed size whenever the inventory position of a stock falls to a pre-specified level (Vollmann et al., 2004). In this paper we manage input buffers of manufacturers and distributors by a FOQ policy with finite lead time and fixed reorder level. The basic quantities of such an inventory management strategy are: the *fixed order quantity* Q, the *lead time*, i.e., the delay between placing an order and receiving the goods in stock; the *demand* D, i.e., the number of units to be supplied from stock in a given time period and the *reorder level* R, i.e., the new orders take place whenever the stock level falls to R.

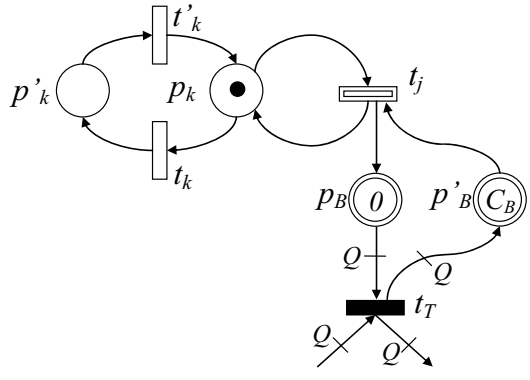
Figure 1(a) shows the FOHPN model for the input buffers (Furcas et al., 2001) managed by the FOQ policy. The continuous place p_B denotes the input buffer of finite capacity C_B and p'_B represents the corresponding available capacity. Thus, at each time instant, with no ambiguity in the notation, we can write $m_B+m'_B=C_B$. We assume that the buffer can receive demands from different facilities and can require the goods from different transporters. Each demand is modelled by a continuous transition t^{D}_i with $i=1,\ldots,m$ so that the demand to be fulfilled is $D_i=\boldsymbol{C}(t^{D}_i)Q'_i$. When $m_B>0$ a transition t^{D}_i with $i\in\{1,\ldots,m\}$ may fire at the firing speed $\boldsymbol{C}(t^{D}_i)=v_i$, reducing the marking of the place p_B with a constant slope $v_iQ'_i$. As soon as m_B falls below the level R_B (or, equivalently, the marking m'_B goes over C_B-R_B) the immediate transition t_1 is enabled. When t_1 fires, place $p_C\in P_d$ becomes marked and performs the choice of the input facility to which new materials/products are requested by enabling one of the transitions t^{T}_i with $i=1,\ldots,n$. If a particular transition t^{T}_i with $i\in\{1,\ldots,n\}$ is selected

and fires after the firing delay $D(t^{T_i}) = \delta_i$, Q_i products are received in the buffer and $C_B - R_B - Q_i$ units are restored in the buffer capacity. Typically, transitions t^{T_i} can represent a transport operation and place p_C selects the transport with minimum transport time among the available ones.

In the following we apply the FOQ policy to different facilities composing the SC and corresponding to input buffers. Note that output buffers are not managed by the FOQ policy since they are devoted just to providing the requested material.



(a)



(b)

Fig. 1. The FOHPN models of an input buffer managed by FOQ policy (a) and a supplier (b).

4.2.2 The supplier module

The supplier is modelled as a continuous transition and two continuous places (see places p_B, p'_B and transition t_j in Fig.1(b)). The continuous transition t_j models the arrival of raw material in the system at a bounded rate v_j that belongs to the interval $v_j \in [V_{j,min}, V_{j,max}]$. We consider the possibility that the providing of raw material is blocked for a certain period.

This situation is represented by a discrete event modelled by two exponentially distributed transitions and two discrete places (p_k and p'_k). In particular, place p_k represents the operative state of the supplier, and $p'_k \in P_d$ is the non-operative state (see Fig.1(b)). The blocking and the restoration of the raw material supply correspond to the firing of transitions t_k and t'_k, respectively. The continuous place p_B models the raw material buffer of finite capacity C_B, and p'_B represents the corresponding available capacity. For the sake of clarity, in Fig. 1(b) we also report transition t_T that, as discussed later (see Section 4.2.4), models the transport operation that corresponds to the withdrawal of material from the buffer.

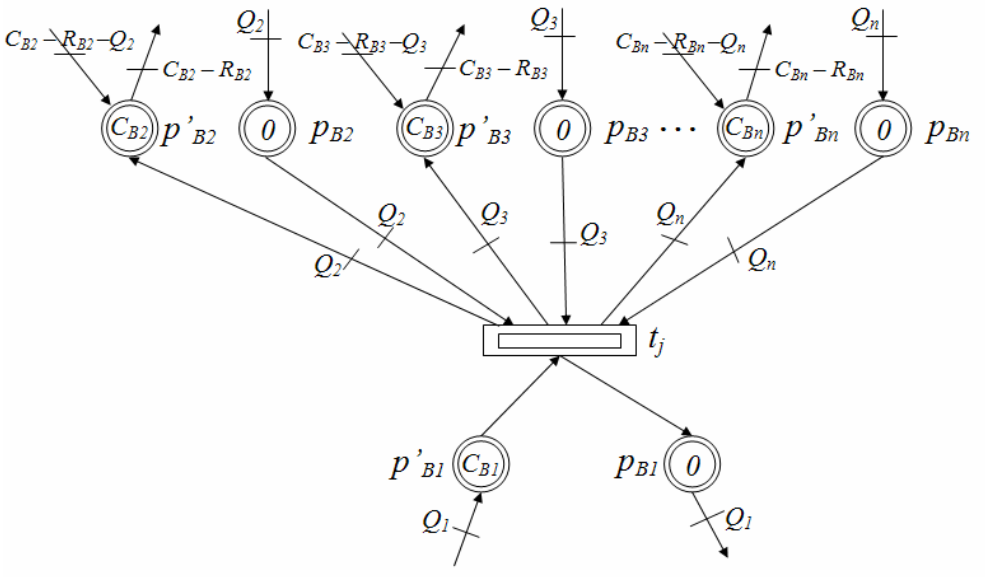


Fig. 2. The FOHPN models of a manufacturer or assembler.

4.2.3 The manufacturer and assembler module

Manufacturers and assemblers are modelled by the FOHPN shown in Fig. 2. More precisely, the continuous places p_{Bi} and p'_{Bi} with i=2,…n describe the input buffers and the corresponding available capacity, respectively. Each buffer stores the input goods of a particular type. Analogously, the continuous places p_{B1} and p'_{B1} model the output buffer. The production rate of the facility is modelled by the continuous transition t_j with the assigned firing speed $v_j \in [V_{j,min}, V_{j,max}]$. Moreover, the firing speed can be optimized according to a given objective function.

4.2.4 The logistic modules

The logistics of SCs are composed by buffers, transporters and distributors.

The buffer modules are described in the inventory management model and in the supplier, manufacturer and assembler modules.

The FOHPN model of transporters is reported in Fig. 3(a). The transporters connecting the different facilities are modelled by a set of discrete deterministic timed transitions t^{T_i} with i=1,…,n. Each transition describes the transport of items of a particular type from a facility to a subsequent one in a constant time interval δ_i=D(t^{T_i}). The control places $p_{C1},\ldots,p_{Cn} \in P_d$ determine the choice of only one type of material to transport among the available set. In addition, place $p_1 \in P_d$ disables the remaining transitions. Moreover, the random stop of the material transport is represented by two places $p_k, p'_k \in P_d$ and two exponentially distributed transitions $t_k, t'_k \in T_E$.

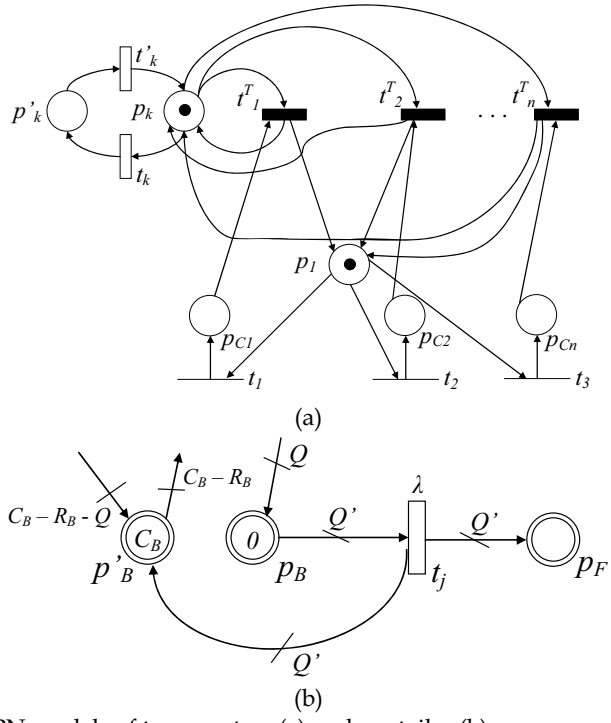


Fig. 3. The FOHPN models of transporters (a) and a retailer (b).

The model of the distributors is represented by an input buffer managed by the FOQ system. Hence, the model is similar to the FOHPN represented in Fig. 1(a) where each continuous transition t^{D_i} , with i=1, …, m, is substituted by a deterministic timed transition representing a transport operation.

4.2.5 The retailer module

The FOHPN model of a retailer is reported in Fig. 3(b). It is a constituted by an input buffer p_B managed by the FOQ policy with a finite lead time and stochastic demand. Hence, the model is similar to the FOHPN represented in Fig. 1(a) where all the continuous transitions t^{D_i} with i=1,…,m are substituted by one or more exponential transitions modelling the stochastic demand of the consumers. Moreover, the continuous place p_F denotes the system output and collects all the products obtained by the retailer.

4.3 An application example of SCs

To illustrate the modelling technique, we consider the SC depicted in Fig. 4 composed by three suppliers S1, S2 and S3, two manufacturers M1 and M2, one distributor D1, two retailers R1 and R2 and eight logistics service providers T1 to T8 that suitably connect the SC facilities. We assume that the system produces a product brand C, ordered by both retailers. Such product is obtained by two manufacturers that receive the input components of type A and B by the suppliers. Moreover, we assume that the SC is managed by the well-known Make To Stock (MTS) policy (Viswanadham & Raghavan, 2000). This means that the system is managed by a push strategy, so that end customers are satisfied from a stock of inventory of finished goods.

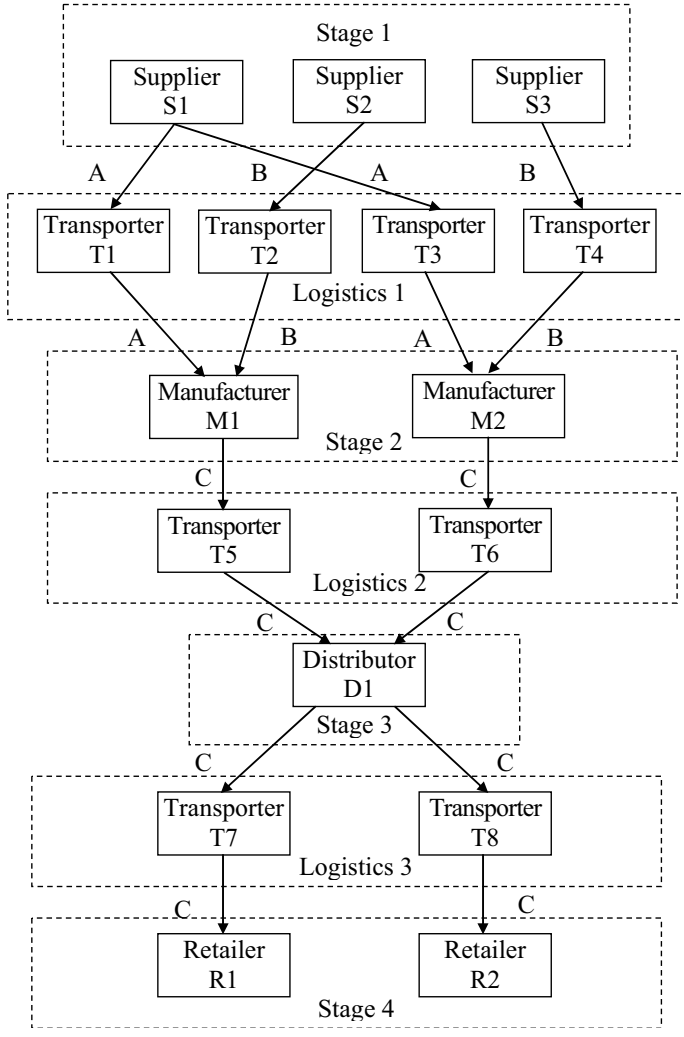


Fig. 4. The SC considered in Section 4.3.

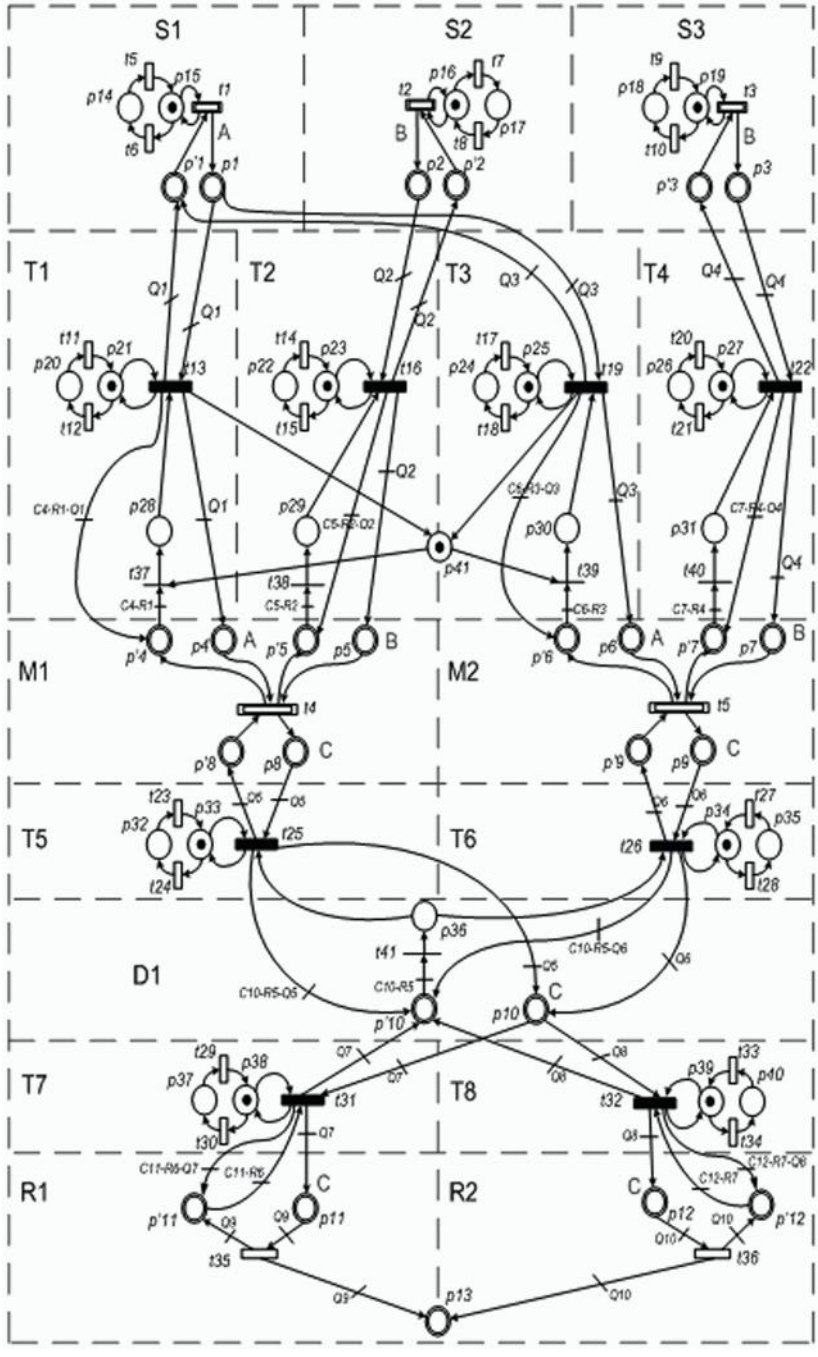


Fig. 5. The FOHPN model of the SC in Fig. 4.

The whole system is modelled by merging all the elementary modules described in the previous section. The resulting FOHPN is reported in Fig. 5, where each facility module is depicted within dashed boxes. The production is determined by the firing of the continuous transitions t_1,t_2,t_3 that describe the input of the raw materials that can be interrupted just by stochastic events. Consequently, under this control technique, each input buffer is managed by the FOQ strategy. Moreover, if the input buffer of manufacturer M1 (M2) requires a particular product, a request is sent to the transporter.

4.3.1 Simulation and optimization

The SC dynamics is analyzed via numerical simulation using the data reported in Table 1, where we can read the manufacturer production rates range, the range of transportation speeds and the average firing delays of discrete stochastic transitions. Table 2 shows further data necessary to completely describe the system, namely the initial markings of continuous places, the buffer capacities for the inventories of each stage and the values of the reorder levels and fixed order quantities.

In order to analyze the SC behavior, some basic performance indices are assumed (Gershwin, 2002, Viswanadham, 2000):

1. the system throughput T, i.e., the average number of products obtained by the retailers in a time unit;
2. the average system inventory SI, i.e., the average amount of products stored in all the system input buffers during the run time TP;
3. the lead time LT=SI/T that is a measure of the time spent by the SC in converting the raw material into final products.

Continuous transitions		Discrete transitions			
	$[V_{min}, V_{max}]$	Exponential	Average firing delay (hours)	Timed	Firing delay (hours)
t_1 t_5 t_7	[2, 4]	t_{22} t_{40}	2	t_{53}	1
t_2 t_3 t_4	[3, 5]	t_{16} t_{26} t_{34}	3	t_{42} t_{43}	2
t_6	[4, 6]	t_{10} t_{14} t_{18}	4	t_{47} t_{48}	2
t_8	[0,7]	t_{24} t_{28} t_{32}	4	t_{52} t_{54}	2
t_9	[0,6]	t_{36} t_{38}	4	t_{44} t_{45}	3
		t_{20} t_{30} t_{41}	5	t_{46} t_{49}	3
		t_{12}	6	t_{50} t_{51}	3
		t_{13}	18		
		t_{21} t_{31}	19		
		t_{11} t_{15} t_{19}	20		
		t_{25} t_{29} t_{33}	20		
		t_{37} t_{39}	20		
		t_{17} t_{27} t_{35}	21		
		t_{23}	22		

Table 1. Firing speeds and average firing delay of continuous and discrete transitions

Initial markings	Product units	Capacities	Reorder levels	Fixed Order quantities
$m_1\ m_5\ m_{11}\ m_{15}$	20	C_1,C_5,C_{11},C_{15}=100	R_1=18	Q_1,Q_6=50
$m_{23}\ m_{25}$	20	C_{23},C_{25}=100	R_2,R_3,R_4=25	Q_2=45
$m_{31}\ m_{37}\ m_{39}$	20	C_{31} =150 C_{37},C_{39}=70	R_5,R_6=15	Q_3=55
$m_3\ m_9\ m_{13}$	15	C_3,C_9,C_{13}=100	R_7,R_8=20	Q_4=40
$m_7\ m_{27}$	25	C_7,C_{27}=100	R_9=30	Q_5=60
$m_{17}\ m_{19}\ m_{29}$	30	C_{17},C_{19},C_{29}=100	R_{10},R_{11}=10	Q_7=30
m_{33}	30	C_{33}=150		Q_8=25
m_{21}	35	C_{21}=100		Q_9=2
$m_{35}\ m_{41}$	0	C_{35}=120		Q_{10}=5

Table 2. Initial marking of odd continuous places, capacities and edge weights.

The FOHPN model has been implemented and simulated in the well-known Matlab environment (The Mathworks, 2006). Indeed, such a matrix-based software appears particularly appropriate for simulating the FOHPN dynamics based on the matrix formulation of the marking update described in Section 3. In particular, the chosen software program is able to integrate modelling and simulation of hybrid systems with the solution of constrained optimization problems, i.e., the IFS vector choice within the set of admissible values by optimizing a particular objective function.

In more detail, after defining the system parameters and the initial marking, the main simulation program first selects the value of each transition timer set point, then determines the set of IFS admissible vectors and solves the optimization program by a suitable Matlab routine; it subsequently determines the next macro-event to occur using an appropriate routine that singles out the enabled transitions. Hence, the simulation determines the next marking with the matrix formulation of the marking update described in Section 3, and finally updates the set point of all transitions so that the next macro-period may be simulated.

All the indices assessing the performance of the SC dynamics are estimated by simulation runs of a time period TP=480 hours and 1000 independent replications. Moreover, the simulations are performed in two operative conditions, denoted *OCi* with i=1,2 and each operative condition *OCi* corresponds to a different choice of the IFS vectors within the set of admissible values.

- *First Operative Condition (OC1).* At each macro-period the IFS vector $\boldsymbol{v}$ is selected so as to maximize the sum of all flow rates (see first item in Section 3.5)
- *Second Operative Condition (OC2).* At each macro-period the IFS vector $\boldsymbol{v}$ is selected so as to minimize the stored volume (see third item in Section 3.5)

The main results of the numerical simulations we carried out are summarized in Table 3. In particular, *OC1* provides the best performances in terms of system throughput. This result is not surprising because in such operational condition the goal was exactly that of maximizing the sum of all flow rates. Moreover, Table 3 shows the average inventories in the two operative conditions. The values show that the SC is able to keep stocks at a satisfactorily high level, so that the demand is satisfied and inventory is not excessive. In particular, as expected, *OC1* corresponds to the highest inventories and *OC2* to the lowest

stocks. Finally, Table 3 reports the obtained lead times in the two conditions, showing that the obtained LT values in *OC1* are greater than those obtained in *OC2*, since the former case corresponds to a higher productivity.

OC1			OC2		
T units/h	SI units	LT hours	T units/h	SI units	LT hours
2.04	1203	589	1.92	817	425

Table 3. The performance indices.

Summing up, a different choice of the production and work rates (as in the two cases *OC1* and *OC2*) let us manage the different performance indices of the SC, i.e. the system may be forced to evolve in an *optimal* way, e.g. while maximizing the flow rates or minimizing the inventory.

5. Conclusions

In this paper we focused our attention on a particular hybrid PN model called FOHPN, that is based on the fluidification of discrete PNs, and whose main feature is that the instantaneous firing speed of continuous transitions keeps constant during each macro-period. In the first part of the paper we discuss in detail the advantages of fluidification, and provide a brief survey of the most important formalisms within the hybrid PN framework. Finally, we showed how FOHPNs can be efficiently used to model SC, and how interesting optimization problems can be solved via numerical simulation, by simply solving on-line a certain number of LPPs.

6. References

Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G. (1995). *Modelling with Generalized Stochastic Petri Nets*, John Wiley & sons, 1995.

Alla, H. & David, R. (1998). "A modeling and analysis tool for discrete events systems: continuous Petri Nets", *Performance Evaluation*, Vol. 33, No. 3, pp. 175-199.

Alur, R., Courcoubetis, C., Henzinger, T.A. & Ho, P.H. (1993). "Hybrid Automata: an Algorithmic Approach to the Specification and Verification of Hybrid Systems", *Lecture Notes in Computer Science*, Springer Verlag, Vol. 736, pp. 209-229.

Amer-Yahia, C., Zerhouni, N., Ferney, M. & El Moudni, A. (1997). "Modelling of Biological Systems by Continuous Petri Nets", *Proc. 3rd IFAC Symp. Modelling and Control in Biomedical Systems*, Warwick, UK.

Amer-Yahia, C. & Zerhouni, N. (2001). " State equation and Stability for a Class of Continuous Petri Nets. Application to the Control of a Production System", *Studies in Informatics and Control*, Vol. 10, No. 4, pp. 301-317.

Andreu, D., Pascal, J.C. & Valette, R. (1996). "Events as a key of a batch process control system", *Proc. CESA'96, Symp. On Discrete Events and Manufacturing Systems*, Lille, France, 1996.

Balduzzi, F., Giua, A. & Seatzu, C. (2001). "Modelling and Simulation of Manufacturing Systems Using First-Order Hybrid Petri Nets", *Int. J. of Production Research*, Vol. 39, No. 2, pp. 255-282.

Balduzzi, F., Giua, A. & Menga, G. (2000). "First-Order Hybrid Petri Nets: a Model for Optimization and Control", *IEEE Trans. Robotics and Automation*, Vol. 16, pp. 382-399.

Beamon, B.M. (1999). "Measuring Supply Chain Performance", *International Journal of Operations and Production Management*, vol. 19, pp. 257-292, 1999.

Bemporad, A., Júlvez, J., Recalde, L., Silva, M. (2004). "Event-driven optimal control of continuous Petri nets", *Proc. 43th IEEE Conf. on Decision and Control*, Atlantis, Paradise Island, Bahamas.

Champagnat, R., Esteban, P., Pingaud, H. & Valette, R. (1998). "Modeling and Simulation of a Hybrid System Through PR/TR PN-DAE Model, *Proc. 3rd Int. Conf. on Automation of Mixed Processes*, Reims, France.

Chen, H. & Hanisch, H.-M. (1998). "Hybrid net condition/event systems for modeling and analysis of batch processes", *Proc. 3rd Int. Conf. on Automation of Mixed Processes*, Reims, France.

Chen, H., Amodeo, L., Chu, F. & Labadi, K. (2005). "Modeling and performance evaluation of supply chains using batch deterministic and stochastic Petri nets", *IEEE Transactions on Automation Science and Engineering*, vol. 2, no. 2, pp. 132-144.

David, R. & Alla, H. (1987). "Continuous Petri Nets", *Proc. 8th European Workshop on Application and Theory of Petri Nets*, Zaragoza, Spain.

David, R. & Alla, H. (2005). *Discrete, continous and hybrid Petri nets*, Springer, Berlin, Heidelberg.

Demongodin, I. & Koussoulas, N.T. (1998). "Differential Petri Nets: Representing Continuous Systems in a Discrete-Event World", *IEEE Trans. on Automatic Control*, Vol. 43, No. 4, pp. 573-579.

Demongodin, I., Caradec, M. & Prunet, F., (1998). "Fundamental Concepts of Analysis in Batches Petri Nets", *Proc. 1998 IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Diego, CA,USA.

Demongodin, I. & Giua, A. (2002). "Some analysis methods for continuous and hybrid Petri nets", *Proc. IFAC World Congress*, Barcelona, Spain

Desrochers, A., Deal, T.J. & Fanti, M.P. (2005). "Complex-Valued Token Petri nets", *IEEE Transactions on Automation Science and Engineering*, vol. 2, no. 4, pp. 309-318.

Dubois, E., Alla, H. & David, R. (1994). "Continuous Petri Net with Maximal Speeds Depending on Time", *Proc. 15th Int. Conf. on Application and Theory of Petri Nets*", Zaragoza, Spain.

Dotoli, M., Fanti, M.P., Giua, A. & Seatzu, C. (2007). "First-order hybrid Petri nets. An application to distributed manufacturing systems", *Nonlinear Analysis. Hybrid Systems*, in press.

Dotoli, M., Fanti, M.P., Meloni, C., & Zhou, M.C. (2005). "A Multi-Level Approach for Network Design of Integrated Supply Chains", *International Journal of Production Research*, vol. 43, no. 20, pp. 4267-4287.

Dotoli, M., Fanti, M.P., Meloni, C. & Zhou, M.C. (2006). "Design and Optimization of Integrated E-Supply Chain for Agile and Environmentally Conscious

Manufacturing", *IEEE Transactions on Systems Man and Cybernetics, part A*, Vol. 36, No. 1, pp. 62-75.

Flaus, J.-M. (1997). "Hybrid Flow Nets for Batch Process Modeling and Simulation", *Proc. 2nd IMACS Symp. On Mathematical Modeling*, Vienna, Austria.

Flaus, J.-M. & Alla, H. (1997). " Structural analysis of hybrid systems modelled by hybrid flow nets", *Proc. European Control Conference*, Brussels, Belgium.

Furcas, R., Giua, A., Piccaluga, A. & Seatzu, C. (2001). "Hybrid Petri net modelling of inventory management systems", *European Journal of Automation APII-JESA*, vol. 35, no. 4, pp. 417-434.

Gaujal, B. & Giua, A. (2004). "Optimal stationary behavior for a class of timed continuous Petri nets", *Automatica*, vol. 40, no. 9, pp. 1505--1516.

Genrich, H.J. & Schuart, I. (1998). "Modeling and verification of hybrid systems using hierarchical coloured Petri Nets, *Proc. 3rd Int. Conf. on Automation of Mixed Processes*, Reims, France.

Gershwin S.B. (2002). "Manufacturing Systems Engineering", Copyright S. Gershwin, Cambridge, MA, USA.

Giua, A. Pilloni, M.T., & Seatzu, C. (2005). "Modeling and simulation of a bottling plant using hybrid Petri nets", *Int. J. of Production Research,* vol. 43, no. 7, pp. 1375-1395.

Giua, A. & Usai, E. (1998). "Modeling hybrid systems by high-level Petri nets", *European Journal of Automation APII-JESA*, vol. 32, no. 9-10, pp. 1209-1231.

Júlvez, J., & Boel R. (2005). "Modelling and controlling traffic behaviour with continuous Petri nets", *Proc. 16th IFAC World Congress*, Prague, Czech Republic.

Júlvez, J., Recalde, L. & Silva, M. (2002). "On deadlock-freeness analysis of autonomous and timed continuous mono-T semiflow nets", *Proc. 41th IEEE Conf. on Decision and Control*, Las Vegas, USA.

Júlvez, J., Recalde, L. & Silva, M. (2003). "On reachability in autonomous continuous {P}etri net systems", *Lecture Notes in Computer Science*, Springer Verlag, vol. 2679, pp. 221-240.

Júlvez, J., Jimènez, E., Recalde, L. & Silva, M. (2004). "Design of observers for timed CPN systems", *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, The Hague, The Netherlands.

Lefebvre, D. (2000). "Estimation of the production frequencies for manufacturing systems", *IMA J. of Management Mathematics*, vol. 11, no. 4.

Mahulea, C., Giua, A., Recalde, L., Seatzu, C. & M. Silva, M., (2006a). "Optimal control of timed continuous Petri nets via explicit MPC", *Lecture Notes in Computer Science*, Springer Verlag, vol. 341, pp. 383-390.

Mahulea, C., Recalde, L. & M. Silva, M. (2006b). "On performance monotonicity and basic servers semantics of continuous Petri nets", *8th Int. Workshop on Discrete Event Systems*, Michigan, USA.

Mahulea, C., Ramirez Treviño, A., Recalde, L., & Silva, M. (2007). "Steady state control reference and token conservation laws in continuous Petri net systems", *IEEE Trans. on Automation Science and Engineering*, in press.

Murata, T. (1989). "Petri Nets: Properties, Analysis and Applications", *Proceedings IEEE*, vol. 77, pp. 541-580.

Petri, C.A. (1962). "Kommunikation mit Automaten (Communication with automata)", Ph.D. Thesis.

Puri, A. & Varaiya, P. (1996). "Decidable Hybrid Systems", *Computer and Mathematical Modeling*, vol. 23, no. 11-12, pp. 191-202.

Silva, M., Teruel, E., & Colom, J. M. (1996). "Linear algebraic and linear programming techniques for the analysis of net systems", *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Lecture notes in Computer Science,* vol. 1491, pp. 309-373, Springer.

The MathWorks Inc., *Matlab Release Notes For Release 14*. Natick, MA, 2006.

Trivedi, K.S. & Kulkarni, V.G. (1993). "FSPNs: Fluid Stochastic Petri Nets", *Lecture Notes in Computer Science*, Springer Verlag, vol. 691, pp. 24-31.

Valentin-Roubinet, C. (1998). "Modeling of hybrid systems: DAE supervised by Petri Nets. The example of a gas storage", *Proc. 3rd Int. Conf. on Automation of Mixed Processes*, Reims, France.

Viswanadham, N. (2000). "Analysis of manufacturing enterprises", Kluwer Academic Publishers, Boston, MA, USA.

Viswanadham, N. & Gaonkar, R.S. (2003)."Partner Selection and Synchronized Planning in Dynamic Manufacturing Networks", *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 117-130.

Viswanadham, N. & Raghavan, S. (2000). "Performance Analysis and Design of Supply Chains: a Petri Net Approach", *Journal of the Operational Research Society*, vol. 51, pp. 1158-1169.

Vollmann, T.E., Berry, W.L., Whybark, D.C. & Jacobs, F.R., *Manufacturing Planning and Control Systems for Supply Chain Management*, Irwin/Mc Graw Hill, New York, 2004.

Zhou, M.C. & Venkatesh, K. (1998). "Modeling, Simulation and Control of Flexible Manufacturing Systems. A Petri Net Approach", World Scientific, Singapore.

6

Modeling and Analysis of Hybrid Dynamic Systems Using Hybrid Petri Nets

Latéfa Ghomri[1] and Hassane Alla[2]
[1]*University Aboubekr Belkaïd*
[2]*University Joseph Fourier*
[1]*Algeria,*
[2]*France*

1. Introduction

Hybrid dynamic systems (HDSs) are currently attracting a lot of attention. The behavior of interest of these systems is determined by the interaction of a continuous and a discrete event dynamics. The hybrid character of a system can owe either to the system itself or to a discrete controller applied to a continuous system. Several works have been devoted to the modeling of HDSs. These topics were tackled from three different angles. The first kind of models are tools initially conceived for continuous systems that were adapted to be able to deal with switched systems. This approach consists of integrating the event aspect within a continuous formalism. Introducing commutation elements in the Bond-graph formalism is an example of this approach. The second kind of models is discrete event systems tools that were extended for HDSs modeling. In this approach, a continuous aspect is integrated in discrete event formalism. An example of such formalism is hybrid Petri nets. The last kind of formalisms are hybrid models, they combine explicitly a discrete event model and a continuous model. The most known model of this category is hybrid automata (HA). This model presents a lot of advantages. The most important is that it combines, explicitly, the basic model of continuous systems, which are differential equations, with the basic model of discrete event systems, which are finite state automata, which facilitate considerably its analysis. The existence of automatic tools for some classes of HA reachability analysis, such as HyTech[1] confer to this formalism a great analysis power. Most verification and controller synthesis techniques use HA as the investigation tool. This makes that the analysis of several hybrid systems formalisms is made after their translation in HA.

In this chapter, we consider the extension of PN formalism, initially a model for discrete event systems, so that it can be used for modeling and control of HDS. The systems studied correspond to discrete event behaviors with simple continuous dynamics. PNs were introduced, and are still used, for discrete event systems description and analysis (Murata, 1989). Currently, much effort is devoted to adapting this formalism so that it can deal with

---

[1] HyTech: *http://www-cad.eecs.berkeley.edu/_tah/HyTech/*

HDSs, and many hybrid PN formalisms were conceived (Demongodin *et al* 1993; Demongodin & Koussoulas, 1998).
The first steps in this direction were taken by David & Alla (1987), by introducing the first continuous PN model. Continuous PNs can be used either to describe continuous flow systems or to provide a continuous approximation of discrete event systems behavior, in order to reduce the computing time. The marking is no longer given as a vector of integers, but as a real number vector. Thus, during a transition firing, an infinitesimal quantity of marking is taken from upstream places and put in the downstream places. This involves that transition firing is no longer an instantaneous operation but is now a continuous process characterized by a speed. This speed can be compared to a flow rate. All continuous PN models defined in the literature differ only in the manner of calculating instantaneous firing speeds of transitions.
From continuous PNs, the hybrid PN formalism was defined by David & Alla (2001), and since it is the first hybrid formalism to be defined from PNs, the authors, simply, gave it the name of hybrid PN. This formalism combines in the same model a continuous PN, which represents the continuous flow, and a discrete T-timed PN (Ramchandani, 1974), to represent the discrete behavior.
We consider in this chapter the extensions of the PN formalism in the direction of hybrid modeling. Section 2 briefly presents hybrid dynamic systems. Section 3 presents the hybrid automata model. In section 4 we discuss continuous Petri nets. These models are obtained from discrete PNs by the fluidification of the markings. They constitute the first steps in the extension of PNs toward hybrid modeling. Then, Section 5 presents two hybrid PN models, which differ in the class of HDS they can deal with. The first one is used for deterministic HDS modeling, whereas the second one can deal with HDS with nondeterministic behavior. Section 6 addresses briefly the general control structure based on hybrid PNs. Finally, Section 7 gives a conclusion and the main future research.

2. Hybrid dynamic systems

A dynamic system is especially characterized by the nature of its state variables. The latter can be of two kinds:

- Continuous state variables are variables defined on a real interval. Time, temperature, pressure, liquid level in a tank…, are examples of continuous variables.
- Discrete variables take their values in a countable set such as natural numbers or Boolean numbers. The state of a valve, the number of parts in a stock, are examples of discrete variables

Figure 1 illustrates the difference between the evolutions of a continuous and a discrete variable as a function of time.
According to the kind of state variables, we can classify the dynamic systems in three categories: continuous systems are systems which exclusively require continuous state variables for their modeling. Discrete event dynamic systems are systems whose modeling requires only discrete state variables. And finally hybrid dynamic systems which are modelled at the same time by continuous state variables and discrete state variables.

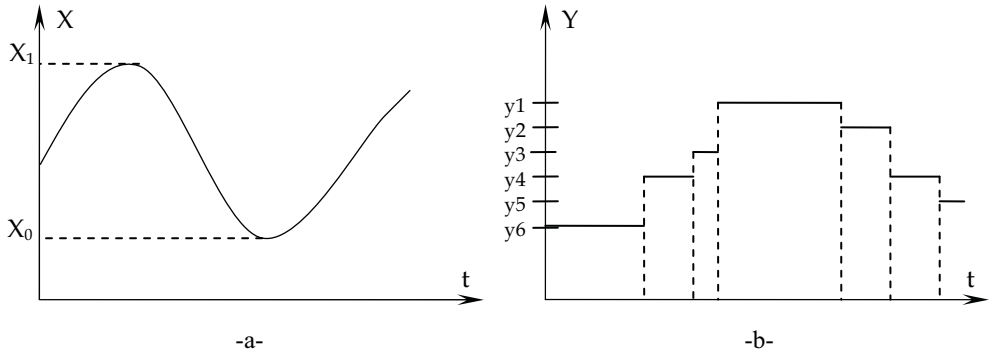


Fig. 1. –a- X is a continuous variable, it takes its values in the real interval $[X_0\ X_1]$. –b- Y is a discrete variable which takes its values in the countable set $\{y_1, y_2, y_3, y_4, y_5, y_6\}$

2.1 Continuous dynamic systems

Chronologically, continuous dynamic systems were the first to be studied. They treat continuous values, like temperature, pressure, flow… etc. The modeling of the dynamic evolution of these systems as a function of time is represented mathematically with continuous models such as: recurrent equations, transfer function, state equations … etc, but the model which is generally used are differential equations of the form:

$$\dot{x} = f(x) \tag{1}$$

Where X is a vector representing the state of the system. The behavior of a continuous system is characterized by the solution of the differential equation $\dot{x} = f(x)$ starting from an initial state x_0.

A continuous dynamic system is said to be linear if it is modelled by a differential equation of the form.

$$\dot{x} = A.x \tag{2}$$

Where A is a constant matrix.

2.2 Discrete event dynamic systems

A discrete events system is described by discrete state variables, which take their values in a countable set. This kind of systems could be either autonomous (not timed) or timed. In the case of an autonomous discrete event system, the variable time is just symbolic, i.e. it is just used to define a chronology between the occurrences of events. In the case of a timed discrete event system, time is explicitly used to define the date of events occurrence. It can be either continuous (dense) or discrete. In the first case, to each event is attached the moment of its occurrence which takes its values in $\Re$, the set of real numbers. In the second case of timed discrete event systems time is only defined on a discrete set. The execution of a sequence of instructions on a processor belongs to this last category, since the executions

may take place only with signals of the processor clock. A discrete event system can be modeled by automata, Petri nets, Markov chains, (max, +) algebra … etc.

2.3 Hybrid dynamic systems

For a long time the automatic separately treated the continuous systems and the discrete event systems. For each one of these two classes of systems exist a theory, methods and tools to solve problems which arise for them. However, the boundaries between the world of continuous systems and that of discrete event systems, are not so clear, the majority of real life systems present at the same time continuous and discrete aspects. Indeed, the majority of the physical systems cannot be classified in one of the two homogeneous categories of the dynamic systems; and state variables of interest may contain simultaneously discrete and continuous variables. In this case the systems are known as hybrid dynamic systems, they are heterogeneous systems characterized by the interaction of a discrete dynamics and a continuous dynamics. The rise of these systems is relatively new, it dates from the 1990s. Figure 2 illustrates the structure of a hybrid dynamic system.

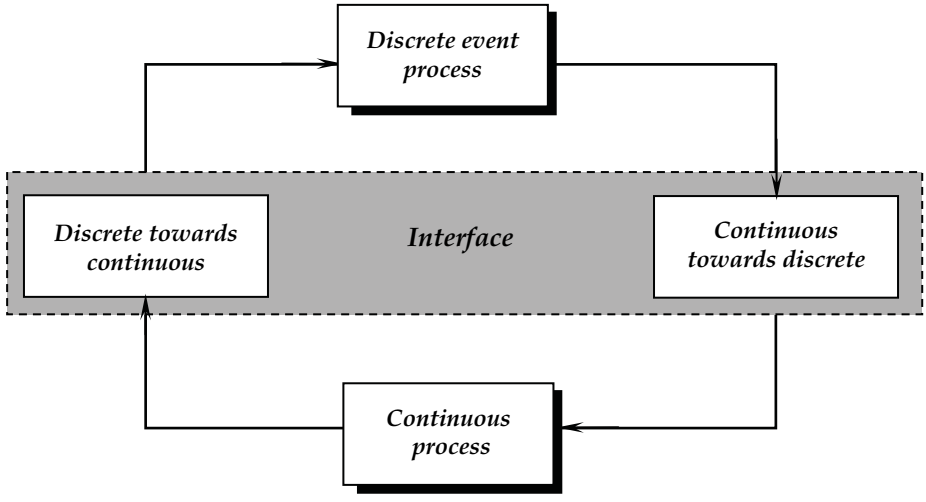


Fig. 2. Structure of a hybrid dynamic system

Research on hybrid dynamic systems is articulated around three complementary axes (Branicky *et al.* 1994; Petterson & Lennartson, 1995): Modeling relates to the formalization of precise models that can describe their rich and complex behavior. Analysis consists in developing tools for their simulation, validation and verification. Control consists in the synthesizing of a discrete (or hybrid) controller on the terms of the performance objectives.
In the sequel, we are interested in a particular class of hybrid dynamic systems; it is the class of continuous flows systems supervised by discrete events systems. This class comprises positive and linear per pieces hybrid systems. A hybrid system is said to be positive if its state variables take positive values in time. And it is said to be linear per pieces if the differential equations describing its continuous evolution are all linear. The particular interest given to the study of this class of systems has two principal reasons. First, it is

sufficiently rich to allow a realistic modeling of many problems. Then, its relative simplicity allows an easy design of tools and models for its description and its analysis. Examples of this class of hybrid systems are given below.

2.4 Illustrative examples

As previously mentioned, a system is said to be hybrid if it implies continuous processes and discrete phenomena. By extension, we can state that physical systems whose certain components vary very quickly (quasi–instantaneously) compared to the others, are also hybrid. A hybrid modeling for this category of physical systems is possible and gives often good results compared to a discrete modeling . We will present two examples of hybrid systems here, the first is a system of tanks implying a (continuous) flow of liquid and the second is a manufacturing system treating a flow of products (discrete dynamics approached by a continuous description).

Example 1: Figure 3 represents a system of tanks. It comprises two tanks which are emptied permanently (except if they are empty) with a flow of 5 and 7 litres/second respectively. The tanks are also supplied in turn, with a valve whose flow is 12 litres/second. The latter has two positions, when it is in position A, it feeds tank 1 and it supplies tank 2 if it is in position B. To commutate between positions A and B the valve needs 0.5 seconds, during which, the valve behaves as if it is in its precedent position.

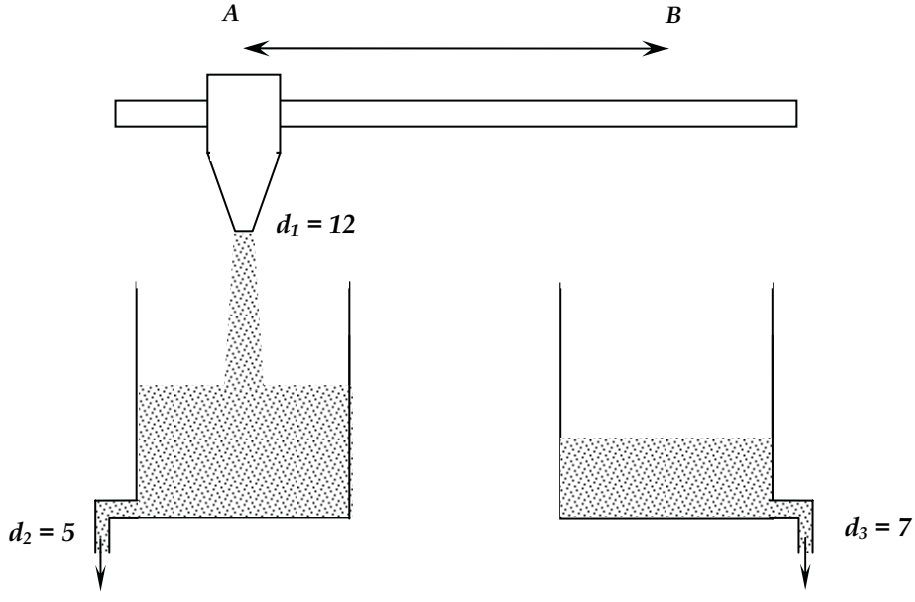


Fig. 3. System of tanks

Example 2: Figure 4 represents a manufacturing system comprising 3 machines and 2 buffers. This system is used to satisfy a periodic request, with a period of 20 time units. Machines 1 and 2 remain permanently operational, while machine 3 can be stopped for the regulation of manufacturing rate. The actions of stopping and starting machine 3 take 0.5 time units. The machines have manufacturing rates of 10, 7, and 22 parts/time units,

respectively. In this system the flow of parts is supposed to be a continuous process, while the state of machine 3 as well as the state of the request is discrete variables.

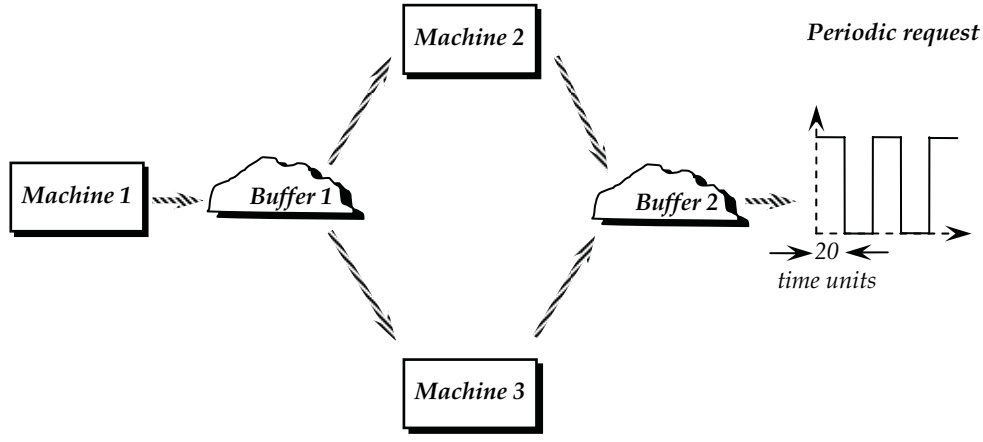


Fig. 4. Manufacturing system

3. Hybrid automata

To integrate the discrete and continuous aspects within the same model, three approaches were presented in the literature. They depend on the dominant model, i.e. the model from which the extension was carried out. We distinguish:

- The continuous approach which consists in integrating the discrete aspect within a continuous formalism. It is an extension of formalisms of continuous systems.
- The discrete approach which consists in integrating the continuous aspect within a discrete events model. The integration of the continuous aspect within the Petri nets model is an example of this approach.
- The hybrid approach which explicitly combines a continuous model and a discrete event model in the same structure. The hybrid aspect is dealt with in the interface between the two parts. An example of such formalisms is hybrid automata that we will present below.

Hybrid automata were introduced by Alur *et al*. (1995) as an extension of finite automata, which associate a continuous dynamics with each location. It is the most general model in the sense that it can model the largest continuous dynamics variety. A HA is defined as follows.

Definition 1 (Hybrid Automata): An n-dimensional HA is a structure *HA = (Q, X, L, T, F, Inv)* such that:

1. Q is a finite set of discrete locations;
2. $X \subseteq R^n$ is the continuous state space; it is a finite set of real-valued variables; A valuation v for the variables is a function that assigns a real-value $v(x) \in R$ to each variable
   $x \in X$; V denotes the set of valuations;
3. L is a finite set of synchronization labels;

4. δ is a finite set of transitions; Each transition is a quintuple T = (q, a, μ, γ, q') such that:
   - $q \in Q$ is the source location;
   - $a \in L$ is a synchronization label associated to the transition;
   - μ is the transition guard, it is a predicate on variables values; a transition can be taken whenever its guard is satisfied;
   - γ is a reset function that is applied when taking the corresponding transition;
   - $q' \in Q$ is the target location;
5. F is a function that assigns to each location a continuous vector field on X; While in discrete location q, the evolution of the continuous variables by the differential equation

$$\dot{x} = f_q(x) \tag{3}$$

   This equation defines the dynamics of the location q;
6. Inv is a function that affects to each location q a predicate Inv (q) that must be satisfied by the continuous variables in order to stay in the location q;

A state of a HA is a pair (q, v) consisting of a location q and a valuation v.

This model present a lot of advantages: It combines, explicitly, the basic model of continuous systems, which are differential equations, with the basic model of discrete event systems, which are finite state automata, this facilitate considerably its analysis; It can model the largest variety of HDSs; It has a clear graphical representation; indeed, the discrete and continuous parts are well identified; The existence of automatic tools for HA reachability analysis, such as HyTech, CMC[2], UPPAAL[3] and KRONOS[4], confer on this formalism a great analysis power. Most verification and controller synthesis techniques use HA as the investigation tool. Several problems, related to analysis of HA properties, could be expressed as a reachability problem. Note that this problem is generally undecidable unless strong restrictions are added to the basic model, to obtain special sub-classes of HA (Henzinger *et al.* 1995). The existence of computer tools allowing the analysis of the reachability problem for some classes of HA makes that the analysis of several hybrid systems formalisms is made after their translation in HA (Cassez and Roux, 2003; Lime and Roux 2003).

4. Continuous Petri nets

Continuous Petri nets were introduced by David and Alla, (1987) as an extension of traditional Petri nets where the marking is fluid. A transition firing is a continuous process and consequently the state equation is a differential equation. A continuous PN allows, certainly, the description of positive continuous systems, but it is also used to approximate modeling of discrete event systems (DES). The main advantage of this approximation is that the number of events occurring is considerably smaller than for the corresponding discrete PN. Moreover, the analysis of a continuous PN does not require an exhaustive enumeration of the discrete state space.

---

[2] CMC: *http ://www.lsv.ens-cachan.fr/_fl/cmcweb.html/*
[3] UPPAAL : *http ://www.uppaal.com/*
[4] KRONOS: *http ://www-verimag.imag.fr/TEMPORISE/kronos/*

As for classical (discrete) Petri nets. We can define two types of continuous Petri nets, namely: autonomous continuous Petri nets and non-autonomous continuous Petri nets.
An autonomous continuous PN allows a qualitative description of continuous dynamic systems, it is defined as follows:
Definition 2 (autonomous continuous Petri Net): An autonomous continuous Petri net is a structure PN = (P, T, Pre, Post, M_0) such that:
1. $P = \{P_1, P_2, \ldots, P_m\}$ is a nonempty finite set of m places ;
2. $T = \{T_1, T_2, \ldots, T_n\}$ is a nonempty finite set of n transitions ;
3. Pre : $P \times T \rightarrow R^+$ is the pre-incidence function that associates a positive rational weight for each arc (T_j, P_i) ;
4. Post : $P \times T \rightarrow R^+$ is the post-incidence function that associates a positive rational weight for each arc (P_i, T_j) ;
5. $M_0 : P \rightarrow R^+$ in the initial marking ;

The following notations will be considered in the sequel:
$^\circ T_J$ is the set of input places of the transition T_J.
T°_J is the set of output places of the transition T_J.
As in a classical PN, the state of a continuous PN is given by its marking; however, the number of continuous PN reachable markings is infinite. That brought David and Alla (2004) to group several markings into a macro-marking. The notion of macro-marking is defined as follows:
Definition 3 (macro-marking): Let PN be an autonomous continuous PN and M_k its marking at time k. M_k may divide P (the set of places) into two subsets:
1. $P^+(M_k)$: The set of places with positive marking ;
2. $P^0(M_k)$: The set of places whose marking is null ;

A Macro-marking is the set of all markings which have the same subsets P^+ and P^0. A macro-marking can be characterized by a Boolean vector as follows:

$$V : \quad P \rightarrow \{0, 1\}$$

$$P_i \rightarrow \begin{cases} 1 \text{ si } P_i \in P^+ \\ 0 \text{ si } P_i \in P^0 \end{cases}$$

The concept of macro-marking was defined as a tool that permits to represent in a finite way, the infinite set of states (markings) reachable by a continuous PN. The number of reachable macro–marking of an n–place continuous PN is less than or equal to 2^n, even if the continuous PN is unbounded, since each macro marking is based on a Boolean state. A macro–marking is denoted m^*_j
Example 3: Let us consider again the hydraulic system of example 1, and consider that the supplying valve is in position A. In this position only the tank 1 is supplied, it is also emptied. While tank 2 is only emptied. The levels of liquid in tanks 1 and tanks 2 are, initially, of H_1 and H_2 respectively.
The continuous PN shown in Figure 5(b) describes the behavior of the system of tanks. Note that the numerical values of the valves flows cannot be represented in an autonomous CPN. The continuous transitions, T1, T2, and T3 represent only a positive flow for the three valves. Places and transitions of the continuous PN are represented with double line to distinguish them from places and transitions of a discrete PN. The firing of transitions T_1, T_2 and T_3 represents material flow through the valves. The marking of places P_1 and P_2 represents quantities of liquid in tank 1 and tank 2

respectively. Figure 5(c) represents the reachability graph; it contains all macro-marking reachable by the continuous PN.

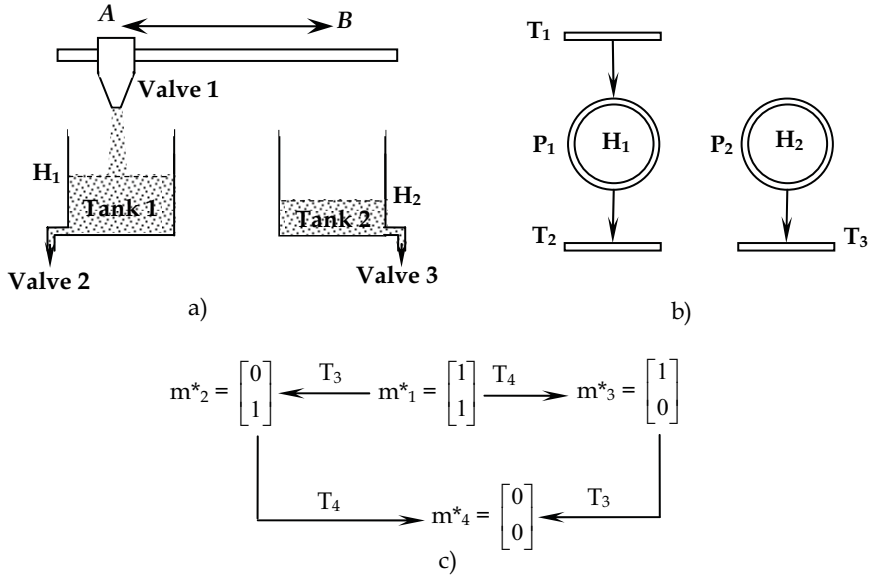


Fig. 5. a) System of tanks, b) Continuous PN describing the system of tanks, c) Reachability graph for the continuous PN

From the basic definition of autonomous continuous PNs, several researchers have defined several timed continuous PNs formalisms. Among these formalisms, we will present the first model to be defined which is always the most studied model, which is constant speed continuous Petri nets. It is defined as follows:

Definition 4 (Constant speed continuous Petri nets): A constant speed continuous Petri net is a structure $PN_C = (PN, V)$ such that:

- PN is an autonomous continuous PN.
- $V: \quad T \rightarrow R^+$
  $T_j \rightarrow V_j$
  is a function that associates to each transition T_j its maximal firing speed V_j.

In a CCPN, a place marking is a real number that evolves according to transitions instantaneous firing speeds. An instantaneous firing speed $v_j(t)$ of a continuous transition T_j can be seen as the flow of markings that crosses this transition. It lies between 0 and V_j for the transition T_j. The concept of validation of a continuous transition is different from the traditional concept met in discrete PNs. We consider that a transition of a CCPN can have two states:

1. The state strongly enabled, if

$$\forall P_i \in {}^\circ T_j, P_i \in P^+$$

   Here, the transition T_j is fired at its maximal firing speed V_j;
2. The state weakly enabled, if

$$\exists P_i \in {}^{\circ}T_j, P_i \in P^0$$

In this case, the transition T_j is fired at a speed v_j lower than its maximum firing speed.
The state equation in a CCPN is as follows:

$$\dot{m} = W.v(t) \quad (4)$$

Where W is the PN incidence matrix. This implies that the evolution in time of the state of a CCPN is given by the resolution of the differential equation (4), knowing the instantaneous firing speed vector. The evolution of a CCPN in time is given by a graph whose nodes represent instantaneous firing speed vectors. Each node is called a phase. In addition, each transition is labeled with the event indicating the place whose marking becomes nil and causes the changing of the speed state. The duration of a phase is also indicated. For more details, see (David and Alla, 2004).

Example 4: Let us consider again the system of tanks, where we associate to each valve its flow rate (figure 6 (a). Moreover, we consider that tank 1 and tank 2 contain initially 70 litres and 36.4 litres respectively. This system is described with the CCPN in Figure 6 (b). The only difference between this model and the autonomous continuous PN in Figure 5 (b) is that with each transition is associated a maximal firing speed.

Since all the places are initially marked, all the instantaneous firing speeds are equal to their maximal value. The marking balance for each place is given by the input flow minus the output flow; then:

At initial time t = 0, $v_1 = 12$, $v_2 = 7$, $v_3 = 3$, then $\dot{m}_1 = 7$ and $\dot{m}_2 = -7$.

Markings m1 and m2 evolve initially according to the following equations, respectively:

$m_1 = 70 + 7.t$

$m_2 = 36.4 - 7.t$

At time t = 5.2 the marking m_2 becomes nil, which defines a new dynamics for the system, as follows:

$v_1 = 12$, $v_2 = 5$, $v_3 = 7$, then $\dot{m}_1 = 7$ and $\dot{m}_2 = 0$.

And after time 5.2, $m_1 = 106.4 + 7.t$ and $m_2 = 0$

This last dynamics is a stationary behavior for the modelled system.

The curves in Figure 7(a) and 7(b) schematize marking m_1 and m_2 dynamics. These plots are made with the software SIRPHYCO[5]. This tool permits the simulation of discrete, continuous and hybrid PNs. The evolution of this model in time can be described thanks to the evolution graph in Fig. 6-c-. It can be noticed that the marking of place P_1 is unbounded while the number of nodes is finite and equal to 2.

5. Hybrid Petri nets

Continuous PNs are used for modeling continuous flow systems; however, this model does not allow logical conditions or discrete behavior modeling (e.g. a valve may be open or closed). For permitting modeling of discrete states, hybrid PNs were defined (David and Alla, 2001). In a hybrid PN, the firing of a continuous transition describes the material flow,

[5] SIRPHYCO: *http://www.lag.ensieg.inpg.fr/sirphyco/*

while the firing of a discrete transition models the occurrence of an event that can, for example, change firing speeds of the continuous transitions.
We find in the literature several types of continuous PN (David and Alla, 2004) and several types of discrete PN integrating time (Ramchandani, 1974; Merlin, 1974). In the autonomous hybrid model definition, there are no constraints on discrete and continuous part types. The most used, which is also the first formalism to be defined, is simply called the hybrid Petri net. It combines a CCPN and a T - timed PN. The combination of these two models confers to the hybrid model a deterministic behavior. It is used for the performance evaluation of hybrid systems.
D-elementary hybrid PNs are another type of hybrid PN formalism. They combine a time PN and a constant speed continuous PN (CCPN) (David and Alla 1987). Time PNs are obtained from Petri nets by associating a temporal interval with each transition. They are used as an analysis tool for time dependent systems.

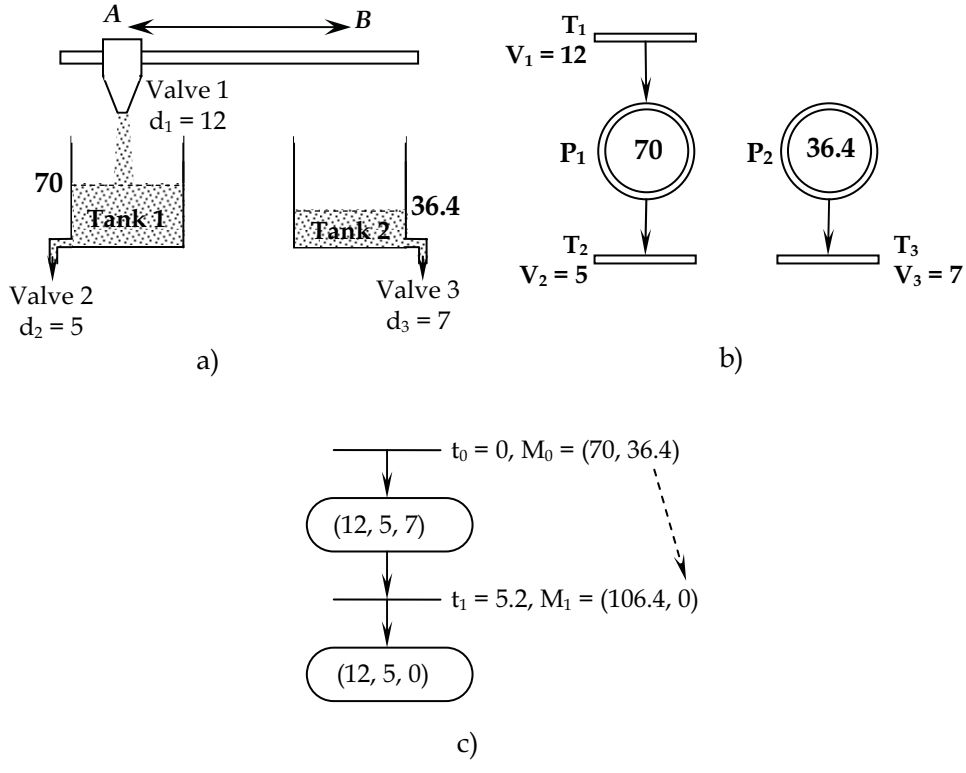


Fig. 6. a) System of tanks, b) Constant speed continuous PN describing the system of tanks, c) the evolution graph for the constant speed continuous PN

However, hybrid PNs were defined before D-elementary hybrid PNs. In order to simplify the presentation, we will start by defining D-elementary hybrid PNs.

5.1 D-elementary hybrid Petri nets

Definition 5 (D-elementary hybrid PNs): A D–elementary hybrid PN is a structure $PN_H = (P, T, Pre, Post, h, S, V, M_0)$ such that:

1. $P = \{P_1, P_2, \ldots, P_m\}$ is a finite set of m places;
2. $T = \{T_1, T_2, \ldots, T_n\}$ is a finite set of n transitions;

We denote PD = {P1, P2,…, Pm'} the set of m' discrete places (denoted by D–places and drawn as simple circles) and TD {T1, T2,…,Tn'} the set of the n' discrete transitions (denoted by D–transitions and drawn as black boxes). PC = P – PD and TC = T – TD denote respectively the sets of continuous places (denoted by C–places and drawn with double circles) and continuous transitions (denoted by C–transitions and drawn as empty boxes).

1. *Pre* : P x T $\rightarrow$ N and *Post* : P x T $\rightarrow$ N are the backward and forward incidence mappings. These mapping are such that:

$$\forall\ (P_i, T_j) \in P^C \times T^D,\ Pre\ (P_i, T_j) = Post\ (P_i, T_j) = 0;$$

$$And: \forall\ (P_i, T_j) \in P^D \times T^C,\ Pre\ (P_i, T_j) = Post\ (P_i, T_j);$$

   This means that no arcs connect C–places to D–transitions, and if an arc connects a D–place P_i to a C–transition T_j, the arc connecting T_j to P_i must exist. This appears graphically as loops connecting D–places to C–transitions.
   These two conditions mean that, in a D–elementary hybrid PN, only the discrete part may influence the continuous part behavior, the opposite never occurs (the continuous part has no influence on the discrete part).
2. *h*: P$\cup$T$\rightarrow${C, D} defines the set of continuous nodes, (h (x) = C) and discrete nodes, (h (x) = D).
3. *S*: $T^D \rightarrow R^+ \times (R^+ \cup \{\infty\})$ associates to each D–transition T_j its firing interval $[\alpha_j, \beta_j]$.
4. *V*: $T^C \rightarrow R^+$ associates a maximal firing speed V_j to each C-transition T_j.
5. M_0 is the initial marking; C–places contain non-negative real values, while D–places contain non-negative integer values.

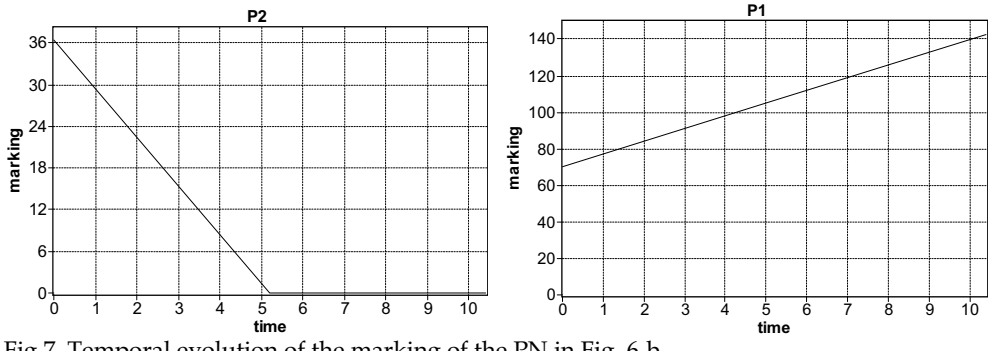


Fig.7. Temporal evolution of the marking of the PN in Fig. 6-b-

Example 5: Consider the system of tanks and suppose that valves 1 may be into the two positions A and B. The passage from position A to position B takes 0.5 seconds, but the commutation decision can be delayed indefinitely for the design of a control. This is why the time interval [0.5 ∞] is associated with the discrete transition T_1. On the other

hand, the passage from position B to position A takes place after exactly 10 seconds from the last commutation (A→B). This is why the time interval [10, 10] is associated with the discrete transition T_2. The D-elementary hybrid PN in Figure 8 describes this hybrid system.

As a D-elementary hybrid PN combines a discrete and a continuous PN, its state at time t is given by the states of the two models. The strong coupling of these models makes it complex to analyze the hybrid model. Translating it into a hybrid automaton permits the use of tools and techniques developed for HA analysis. Ghomri *et al.* (2005) developed an algorithm permitting translation of a D-elementary hybrid PN into a HA. In the sequel, we briefly present this algorithm.

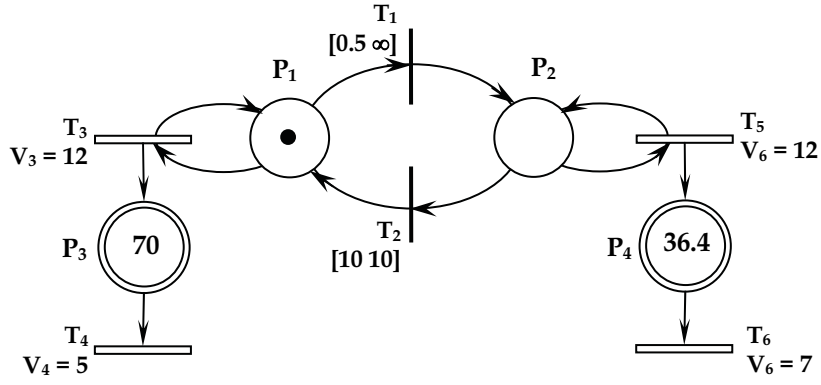


Fig.8. D-elementary hybrid Petri net describing the system if tanks

5.2 Translating D-elementary hybrid Petri nets into hybrid automata

It is, generally, very complex to translate a hybrid PN into a hybrid automaton because of the strong coupling between discrete and continuous dynamics. D-elementary hybrid PNs represent only a class of hybrid PNs, which permits modeling of frequently met actual systems: i.e. the class of continuous flow systems controlled by a discrete event system. The translation algorithm consists in separating the discrete and the continuous parts. Then, the translation into an automaton is performed in a hierarchical way. The algorithm is based on three steps as follows:

1. Isolate the discrete PN of the hybrid model and construct its equivalent timed automaton. Locations of the resulting timed automaton are said macro-locations.
2. Construct the hybrid automaton corresponding to each macro-location of the timed automaton resulting from the previous step.
3. Replace transitions between macro-locations by transitions between internal locations.

We detail these three steps through the following example.

Example 6: Consider the D-elementary HPN in Figure 8. Its discrete part is set again in Figure 9(a). The timed automaton corresponding to this time PN is represented in Figure 9(b).

To each location of the timed automaton, corresponds a marking of the time PN, and therefore a configuration of the CCPN. For instance, if P_1 is unmarked, T_3 may be

eliminated from the CCPN in figure 8. The location S_2, for example, corresponds to the time PN marking vector $[m_1\ m_2]^T = [0\ 1]^T$, for which the continuous part is reduced to CCPN in Figure 10(a). This CCPN may be translated into the HA in Figure 10(b).

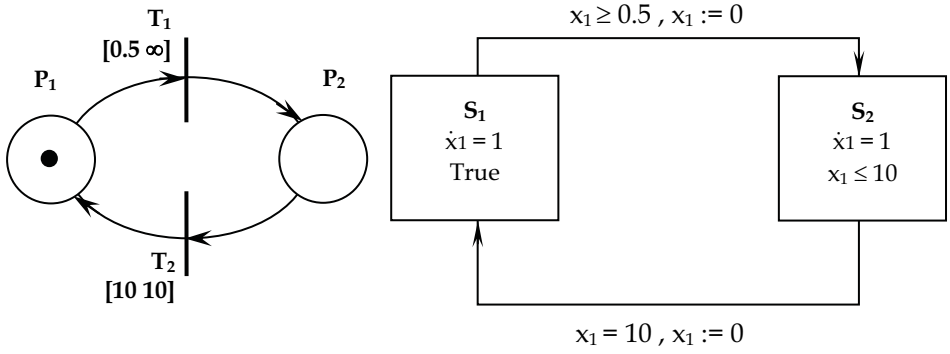


Fig.9. time Petri net and its equivalent time automaton

After the second step of the translation algorithm, we obtain a hierarchical form of a HA, formed from macro-locations each containing a HA describing the continuous dynamics in it. A generic representation of the model resulting after step 2 of the algorithm is given in Figure 11.

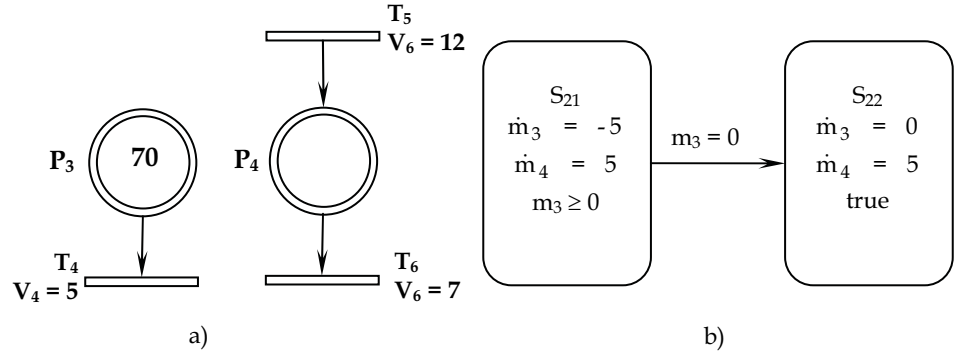


Fig.10. Constant speed continuous Petri net and its equivalent hybrid automaton

The location number of the resulting hybrid automaton depends on two parameters: (i) the location number of the TA describing the discrete part behavior, denoted as n; (ii) the continuous place number of the continuous part, denoted as m. The first parameter n is finite for a bounded time PN; although the propriety of boundedness is undecidable for a time PN, there exist restrictive sufficient conditions for its verification (Berthomieu and Diaz 1991). This first parameter defines the macro-location number. The second parameter m defines the number of locations inside a macro-location. As mentioned before, we can always model the behavior of a continuous PN by a HA with a finite number of locations,

even if the continuous PN is unbounded; this number is least or equal to 2^m. We have therefore a resulting HA that contains at the most ($n.2^m$) locations. This is an important result since it is generally impossible to bound a priori the number of reachable states in a hybrid PN.

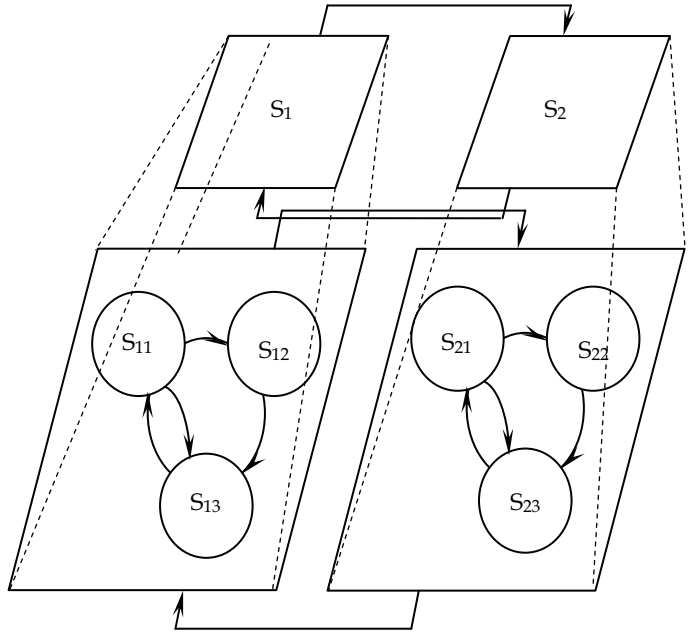


Fig.11. Generic schematization of model resulting from the second step of the algorithm

5.3 Hybrid Petri nets

A hybrid PN is distinguished from a D-elementary hybrid PN by the fact that the former contains a T-timed PN for modeling the discrete part—timed fixed values are associated with each transition—whereas the latter model contains a T-timed PN.

Definition 6 (hybrid Petri Net): A hybrid PN is a structure PN_H = (P, T, Pre, $Post$, h, S, V, M_0) such that :

1. $P = \{P_1, P_2, \ldots, P_m\}$ is a finite set of m places. $P = P^D \cup P^C$;
2. $T = \{T_1, T_2, \ldots, T_n\}$ is a finite set of n transitions. $T = T^D \cup T^C$;
3. $Pre : P \times T \rightarrow N$ and $Post : P \times T \rightarrow N$ are the backward and forward incidence mappings.

These mapping are such that:

$$\forall (P_i, T_j) \in P^D \times T^C, \text{Pre}(P_i, T_j) = \text{Post}(P_i, T_j);$$

1. $h : P \cup T \rightarrow \{C, D\}$ defines the set of continuous nodes, (h (x) = C) and discrete nodes, (h (x) = D).
2. $S : T^D \rightarrow Q^+$ associates to each D-transition T_j a duration d_j.
3. $V : T^C \rightarrow R^+$ associates a maximal firing speed V_j to each C-transition T_j.

4. M_0 is the initial marking.

The condition on backward and forward incidence mappings means that, if an arc connects a D-place P_i to a C-transition T_j , the arc connecting T_j to P_i must exist. And vice versa. This appears graphically as loops connecting D-places to C-transitions. It means that a discrete token cannot be split by a continuous transition. The hybrid PN model, as defined below, allows modeling of the logical conditions, but it allows also the modeling of the transformation of a continuous flow into discrete parts and vice versa.

Example 7: Let us consider again the system tanks, and suppose that we have the following control strategy: we want to keep the liquid levels in tank 1 at least than a fixed level H_{max}. The hybrid PN in Figure 12 describes a system that satisfies this specification on the level in tanks.

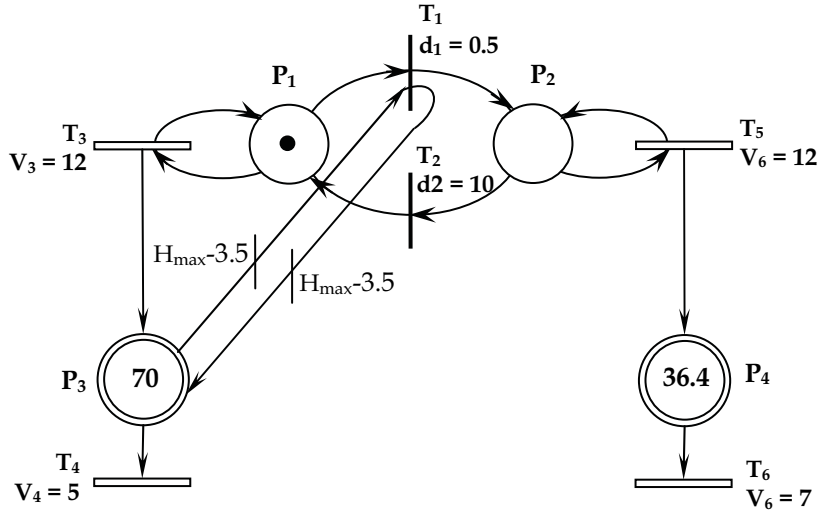


Fig.12. Hybrid Petri net describing the system of tanks with a restriction on its marking.

The weights (H_{max} – 3.5) associated with the arcs correspond to the minimal thresholds of tank 1 taking into account the delay 0.5.

6. Controller synthesis

The controller synthesis of HDS drifts directly from Ramadge and Wonham (1989) theory. They synthesize, from a discrete event system, a controller whose role is to forbid the occurrence of certain events. The controller decision to forbid an event depends only on the past of the system, i.e. of events, which already occurred. The aim is that the system coupled to its controller respects some given criteria.

Many researches were devoted to the problem of controller synthesis autonomous discrete event systems. This problem is thus well solved for this category of systems. The number of works relating to real time system controller synthesis is also very significant (Altisen *et al.* 2005). However, few works were devoted to solving this problem for the HDS (Wong-Toi, 1997; Antsaklis *et al.* 1993; Lennartson *et al.* 1994; Peleties & DeCarlo, 1994).

The controller synthesis of a dynamic system (autonomous, timed or hybrid) is generally based on three steps:

1. the behavioral description of the system (called an open loop system) by a model;
2. the definition of specifications required for this behavior;
3. the synthesis of the controller which restricts the model behavior to the required one, using a controller synthesis algorithm.

These algorithms consider the open system S and the specification on its behavior ϕ. and try to synthesize the controller C so that the parallel composition of S and C(S || C) satisfies ϕ. These algorithms use traditionally automata (finite state automata, timed automata and hybrid automata) because of their ease of formal manipulation; however, a model like HPN is preferred in the first step (the step of behavior description).

Consider an open loop Hybrid system; the aim of controller synthesis is to construct a controller that satisfies the specifications for the closed loop hybrid system. These specifications imply, generally, restrictions on the closed loop hybrid system. They can be either (1) specifications on the discrete part (this type of specification forbids certain discrete states); or (2) specification on the continuous part; in this case the specification has the form of an invariant that the continuous state must satisfy. This implies that the continuous state of the closed loop hybrid system is restricted to a specified region. The open problem is synthesizing the guards associated with the controllable transitions so that the specifications are respected leading to a maximal permissive controller.

7. Conclusion

Some extensions of PNs permitting HDS modeling were presented here. The first models to be presented are continuous PNs. This model may be used for modeling either a continuous system or a discrete system. In this case, it is an approximation that is often satisfactory.

Hybrid PNs combine in the same formalism a discrete PN and a continuous PN. Two hybrid PN models were considered in this chapter. The first, called the hybrid PN, has a deterministic behavior; this means that we can predict the occurrence date of any possible event. The second hybrid PN considered is called the D-elementary hybrid PN; this model was conceived to be used for HPN controller synthesis.

Controller synthesis algorithms consider the open system S and the specification on its behavior ϕ and try to synthesize the controller C, so that the parallel composition of S and C(S || C) satisfies ϕ. These algorithms use traditionally automata (finite state automata, timed automata and hybrid automata) because of their ease of formal manipulation; however, this model is not the most appropriate for behavior description. For coupling the analysis power of hybrid automata with the modeling power of hybrid PNs, an algorithm permitting translation of D-elementary hybrid PNs into hybrid automata was presented. Our future research aim is to generalize the existing results to the control of hybrid systems modeled by hybrid PNs.

8. References

Altisen, K. ; Bouyer, P. Cachat, T. Cassez F. & Gardey G. (2005) Introduction au contrôle des systèmes temps-réel, Proceedings of modélisation des systèmes réactifs, (MSR'05), France

Alur, R. Courcoubetis, C. Halbwchs, N. Henzinger, T.A. Ho, P. H. Nicolin, X. Olivero, A. Sifakis J. & Yovine, S. (1995) The Algorithmic Analysis of Hybrid Systems, *Theoretical computer science*, Vol. 138, pp.3-34

Antsaklis, P.J. Stiver, J.A. Lemmon, M.D. Hybrid system modeling and autonomous control systems, in: R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel, (1993) Hybrid Systems, in: Lecture Notes in Computer Science, vol. 736, Springer-Verlag, pp. 366–392.

Berthomieu, B. & Diaz, M. (1991) Modelling & verification of time dependent systems using time Petri nets, IEEE Transactions on software engineering, vol. 17(3), pp 259-273

Branicky, M. Borkar, V. Mitter, S. (1994) A unified framework for hybrid control, in proceedings of IEEE Conference and Decision and Control, CDC, Lake Buena Vista, USA, pp. 4228–4234.

Cassez, F.; & Roux, O.H. (2003) Traduction Structurelle des Réseaux de Petri Temporel vers les Automates temporisés, *Proceedings of 4ième colloque Francophone sur la modélisation des systèmes réactifs, (MSR'03)*, Metz, France

David, R.; & Alla, H. (2001) On Hybrid Petri Nets. *Discrete Event Dynamic systems: Theory & Applications*. vol. 11, pp. 9-40

David, R.; & Alla, H. (2004) *Discrete, Continuous, & Hybrid Petri Nets,* Springer

David, R.; Alla, H. (1987) Continuous Petri Nets, *Proceedings of the 8th European Workshop on Application & Theory of Petri Nets*, Saragossa, Spain, pp. 275-94

David, R.; Alla, H. Autonomous, (1990) Timed and Continuous Petri Nets, *Proceedings of the 11th Int. Conf. on Application & Theory of Petri Nets,* Paris, France, pp. 367-386

Ghomri, L. Alla H. & Sari, Z. (2005) Structural & hierarchical translation of hybrid Petri nets in hybrid automata, *Proceedings of IMACS'05*, Paris, France

Henzinger,; T.A. Kopke, P.W. Puri, A. & Varaiya. P. (1995) What's decidable about hybrid automata? *Proceedings of* 27th *Annual Symposium on Theory of Computing*, ACM Press, 373-382

Lennartson, B. Egardt, B. Tittus, M. (1994) Hybrid systems in process control, in: Proc. of the 33rd CDC, Orlando, FL, USA, 1994, pp 3587–3592

Merlin, P.; (1974) A study of the recoverability of computer system, PhD thesis, Dep. comput. Sci, Univ. California, Irvine,

Murata, T. (1989) Petri-nets: properties, analysis and applications, in proceedings of IEEE 77 (4) 541–580.

Peleties, P. DeCarlo, R.A. (1994) A modeling strategy for hybrid systems based on event structure, Discrete Event Dynamic Systems: Theory and Applications 30 (9) 1421–1427

Pettersson, S. Lennartson, B. (1995) Hybrid modelling focused on hybrid Petri nets, in: 2nd European Workshop on Real-Time and Hybrid Systems, Grenoble, France, pp. 303–309.

Ramadge, J.G.; & Wonham, W.M. (1989) The Control of Discrete Event Systems, *Proceedings of the IEEE*, vol. 77(1), 81-97

Ramchandani, C., (1974) Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. PhD thesis, MIT, Cambridge, Feb

Wong-Toi, H.; (1997) The synthesis of controllers for linear hybrid automata. *In Proceedings of the 36th Conference on Decision & Control. IEEE* Computer Society Press 4607 – 4612

7

Use of Petri Nets for Modeling an Agent-Based Interactive System: Basic Principles and Case Study

Houcine Ezzedine and Christophe Kolski
LAMIH, University of Valenciennes
France

1. Introduction

Several architecture models of interactive systems have been put forward by researchers over the past twenty years. Two main types of architecture can be distinguished: architectures with functional components: Langage (Foley & Vandam, 1982), Seeheim (Pfaff, 1985) and ARCH (Bass et al., 1991) and architectures with structural components: PAC (Coutaz, 1987), PAC-Amodeus (Nigay, et al., 1997), MVC (Goldberg, 1983), AMF (Tarpin-Bernard & David, 1999), H^4 (Guittet, 1995),…. The approaches currently used in interactive system design adopt a modular structuring aimed towards a better apprehension of the reactivity, flexibility, maintainability and re-use. Agent-based approaches are promising in this way.

In the agent-based architecture proposed, we suggest using a division into three functional components: the *application agents* which handle the field concepts and cannot be directly accessed by the user; the *interactive agents* (or *interface agents*, or presentation agents) which , unlike the application agents, are in direct contact with the user (they can be seen by the user); the *dialogue control agents* which are also called mixed agents (Ezzedine & Trabelsi, 2005). Each agent therefore plays a role within its group; this role can be expressed in the form of the services it offers in the interactive system.

We use so-called agent Petri Nets (PN) to model a priori the services offered by each interface agent: a service is defined as being a quadruplet S ={E, C, R, P}, with E: the event which triggers the service, C: the conditions to be met in order to perform this service, R: the resources necessary for the service to be performed, P: the property of this service, which can be either an operation concerning the agent alone (with or without a change of state for the interactive agents), or a call for the service of another agent. The succession of various calls for services gives rise to the succession of page-screens in the human-computer interface.

This chapter begins with a state of the art about the use of Petri nets in Human-Machine Interaction. Then we explain the problem relating to agent-based architectures of interactive systems, and we propose a solution for the modeling of the interface agents of such architectures. Lastly, we illustrate our approach by a case study.

2. Use of PN in the field of human-computer interaction

Petri nets allow the modeling and visualization of behaviors comprising parallelism, synchronization and resource sharing. Their power lies in their formal aspect; they allow the modeling of discrete systems evolving in parallel, which represents a great contribution for the modeling of various facets relating to human-machine interactions, particularly in complex systems. Various complementary approaches are found on this subject in the literature, based on the use of various types of PN in the field of human-machine interaction, a subject which has been developed ever since the end of the Eighties (Williem & Biljon, 1988). They are given here in a list which is not intended to be exhaustive, but rather to be representative.

Thus, PN were used: (1) before design phases (with an aim of human-computer interaction specification), for task modeling (also called prescribed or theoretical task); it corresponds to the task, envisaged by the designer(s), to be carried out by the user and/or the machine (*a priori* modeling); (2) for the modeling of the human activities (*a posteriori* modeling); this modeling follows the phase of evaluation of the interactive system in a real or simulated situation with users. By a confrontation of the *a priori* and *a posteriori* models, and by an analysis of the differences between these two complementary sets of PN, it is possible to detect design errors, lacks of information, and so on, and to put forward proposals, especially concerning the improvement of human-computer interaction (Abed, 1990; Abed et al. 1992; Abed, 2001). Figure 1 shows an example of modeling with PN of the human-machine interaction planned for part of an interactive application relating to a post of transport network supervision (Ezzedine et al., 2003). The places in the PN represent the actions carried out by the user whereas the transitions represent the reactions from the user interface; the graphic components present on it (bottom left part on figure 1) rise directly from the elements described on the PN (right part of figure 1).

Very important basic research was undertaken by P. Palanque, R. Bastide and their colleagues on the use of the PN for the checking and validation of interactive systems (Palanque & Bastide, 1990; Palanque et al., 1995; Navarre et al., 2003; Winkler et al., 2006…). For instance, they proposed rule-based mechanisms for the automatic evaluation of PN-based models of interactive systems (Palanque et al., 1999).

In the works on the ICO (Interactive Cooperative Objects) (Palanque, 1992; Palanque 1997) and the TOOD method (Task Object Oriented Design) (Mahfoudhi et al., 1995; Tabary & Abed, 2002), Object Petri nets are used to model human tasks in a HCI context and to specify and then design object oriented interactive systems.

Ezzedine and Kolski (2005) present a method for the modeling of cognitive activity also using object Petri nets: the method includes the recognition of the various classes of situation (normal and abnormal) which human operators are likely to meet whilst performing their tasks; each of these classes is described according to the characteristics of the state of the system (Kaddouri et al., 1995).

From a description of normal and abnormal situations possible in process control applications, Moussa et al. (2002, 2006) use interpreted Petri Nets for human-machine dialogue specification.

Kontogianis (2003) chooses to use colored Petri nets for ergonomic task analysis and modeling with emphasis on adaptation to system changes. Gomes et al. (2001) propose an interesting approach based on reactive Petri nets (inherited from colored Petri nets) for human-machine interface modeling.

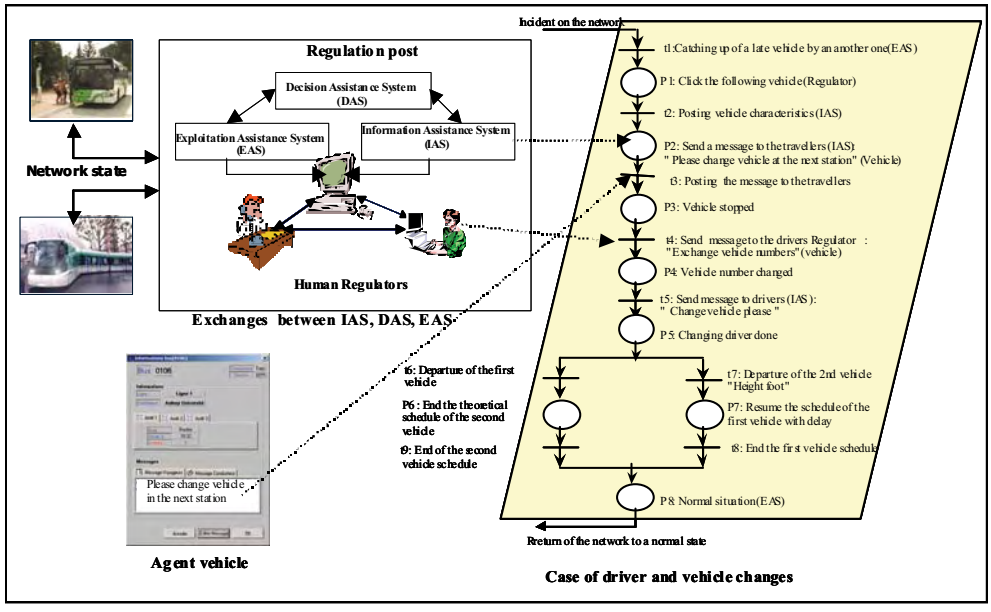


Fig. 1. Human-computer interaction modeling using PN

Bernonville et al. (2006) propose a method to facilitate the re-engineering of existing interactive software by proposing a common framework for Software Engineers and Human Factor specialists: their method explicitly combines Petri Nets and ergonomic criteria.
To our knowledge, none of these works is interested in modeling the agents which make up agent-based interactive systems, by establishing a direct link with the software architecture.

3. Problem of modeling related to agent-based architectures of interactive systems

The architecture of a computer system is a set of structures, each including: components, outside visible properties of these components and relations which the components maintain (Bass et al., 1991). We are only interested in interactive systems: in this context, the architecture models aim to provide a framework for the design and the realization of the complete system, emphasizing clearly the part with which the user interacts. Existing architectures break up the interactive system into modules and define specific roles for each module, contributing to the correct execution of the complete system. Two main types of architecture can be distinguished: architectures with functional components (Langage, Seeheim and Arch) and architectures with structural components (PAC, PAC-Amodeus, MVC…). It should also be noted that certain classifications emphasize three categories (centralized models, distributed or agent-based model, hybrid models).
The classic models of interactive systems distinguish three essential functions *(presentation, control* and *application)*. Some models, such as the Seeheim (Pfaff, 1985) and ARCH models, consider these three functions as being three distinct functional units. Other approaches using structural components, and in particular those said to be distributed or agent

approaches, suggest grouping the three functions together into one unit, the agent. The agents are then organised in a hierarchical manner according to principles of composition or communication: for example PAC (Coutaz, 1997) or its variants, or the MVC model (Model-View-Controller) of Smalltalk and its recent evolutions (Goldberg, 1984), AMF and its variants (Ouadou, 1994), H⁴ (Guittet, 1995)…

These architecture models preach the same principle based on a separation between the system (application) and the human-computer interface (HCI). Thus, an architecture must separate the application and the HCI, define a distribution of the services of the interface and define a protocol of information exchange. One of the interests in separating the interface and the application is to make it easier to modify the interface without changing the application (Coutaz, 1997).

The architecture adopted can be considered as being intermediate as it borrows elements for its principles from both types of model given above at the same time whilst being functional and structural (Ezzedine et al., 2001). In (Ezzedine et al., 2003) and (Ezzedine et al., 2005), we proposed an architecture ensuring separation in three functional components, which we called respectively: *interface with the application* (connected to the application), *dialogue controller* and *presentation* (this component is directly linked to the user), figure 2.

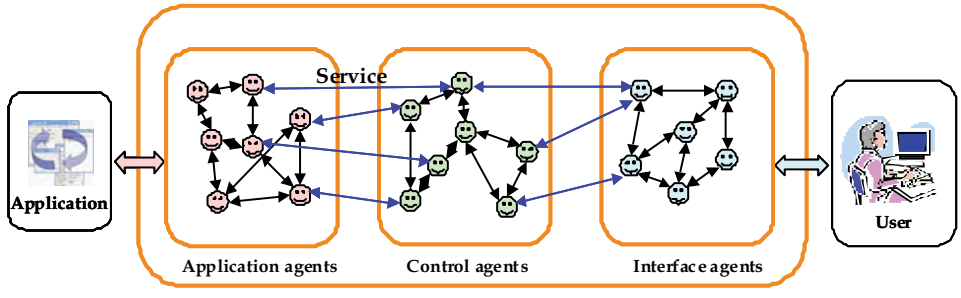


Fig. 2. Agent-based Architecture of interactive system

These three components group together agents:

- the *application agents* which handle the field concepts and cannot be directly accessed by the user. One of their roles is to ensure the correct functioning of the application and the real time dispatch of the information necessary for the other agents to perform their task;
- the *control* (or *dialogue controller) agents* which are also called mixed agents; these provide services for both the application and the user. They are intended to guarantee coherency in the exchanges emanating from the application towards the user, and vice versa;
- the *interactive agents* (or *interface agents*, or *presentation* agents); unlike the application agents, these are in direct contact with the user (they can be seen by the user). These agents co-ordinate between themselves in order to intercept the user commands or requests, and to form a presentation which allows the user to gain an overall understanding of the current state of the application. In this way, a window may be considered as being an interactive agent in its own right; its specification describes its presentation and the services it has to perform.

In the following part, we explain how the agents which make up such agent-based interactive systems are modeled. We focus on the interface agents.

3. Principles of modeling by PN of interactive systems with agent-based architecture

Initially, we will point out the basic principles of the parametrized Petri nets which inspired our modeling approach. Then we will explain how the services of the various interface agents of interface can be modeled.

3.1 Parametrized Petri nets

Parametrized Petri nets are classified amongst the high level PN. They allow the modelling of dynamic systems (Gracanin et al., 1994) according to the following principle: it is possible to link, in one parameter of these PN, a coherent set of objects or of values taken by the objects. This makes it possible to handle sets of objects, thus reducing the complexity of the representation.

A parameterized Petri net is a n-tuple: (C, D, Pp, T, I, O) where:

- C: the set of the values of the parameters; a parameter is a class of objects or values taken by the objects.
- D: the set of all the vectors built starting from the values. In such a network, in fact the vectors are produced or consumed (tokens).
- Pp: the set of the places of the network, called parameterization descriptor.
- T: the set of all the transitions from vectors representing all the actions which can be carried out by the system.
- I: the set of consumed tokens (input)
- O: the set of produced tokens (output)

3.2 Modeling of interface agents according to a set of services

The concept of an agent's service such as it was introduced by (Maoudji et al., 2001) considers the service as an action which involves the agent itself or other agents. For a service to be started, the agent needs:

- the appearance of the trigger event
- the checking of the condition in connection with the service,
- the checking of some resources which may be necessary for the establishment of the service.

Formally the service is defined by a quadruplet (Moldt et Wienberg, 1997), Figure 2, Service={E, C, R, P} where:

- E: the event which triggers the service, for example a user action or a display of an alarm or anomaly message coming from an application.
- C: a condition to check in order to carry out the service, such as the exceeding of a threshold as regards the value of one of the application's variables, or the presence of any risk for the application
- R: the resources necessary for the service to be performed.
- P: the property of the service which can be internal (the resulting action affects the agent itself) or external (the resulting action relates to other agents); the latter will be interpreted as an event.

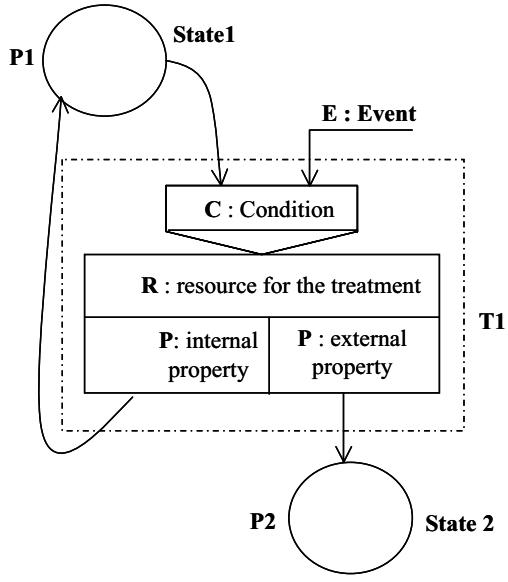


Fig. 2. Modeling of an agent's service (Maoudji et al., 2001)

3.3 Concept of PN used to model the agents of the interactive system (Agent PN)

Taking the concept of service of an agent introduced by (Maoudji et al., 2001) as a starting point, we propose the modeling of the behaviour of an agent by the modeling of the set of its services, Figure 3. The services of each agent can be represented by external actions on

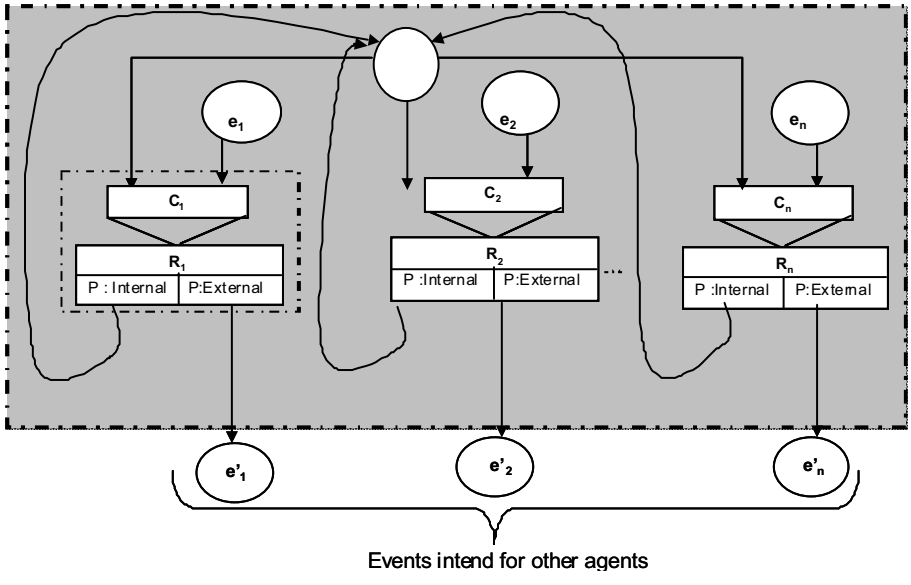


Fig. 3. Modeling of agent's services with agent PN (Ezzedine et al., 2001)

other agents; they are then interpreted as trigger events for other agents such as e'_1, e'_2 and up to e'_n, or by actions on the agents themselves such as looping on the place which is at the entry of their condition C. The service of an agent can be achieved by the execution of an internal or external action only if the condition C is true: i.e. if the place upstream of the condition C is marked (and thus has at least one token), and if event e_i is triggered and the resource necessary to achieve the action is available.
We distinguish three types of place in the PN agent which are:

1. An *agent* place*:* which always contains the current view of the agent (result of a service).
2. An *event* place*:* which contains an event which triggers the service.
3. An intermediate place*:* which contains the events bound for another agent.

If we model an interface with n agents and m services for each agent, we will obtain a less readable PN agent, because of this and in order to mitigate the problem of the complexity of the PN, we increase the model by the capacity to abstract the set of services of an agent in one single form, Figure 4.

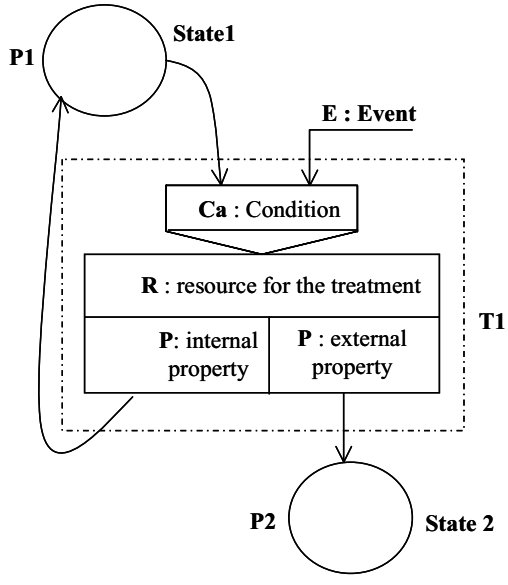


Fig. 4. Abstraction of the services of an agent.

Formally, an agent (set of n services) is defined by a quadruplet as follows:
Agent={E, C, R, P} where:

- E= {e_1, e_2,..., e_i,…, e_n}, set of events which trigger agent services (**n** events for **n** services). An event is characterized by two fields: its identifier and the moment of the appearance. For example, an alarm may be triggered when the threshold of a variable to be supervised in an industrial process is exceeded (temperature too high in a chemical process, late bus in a transport system…).
- C= {c_1, c_2,..., c_i, ..., c_n}, set of conditions necessary for the establishment of the services. Each service must check a condition so that it can trigger itself off. For example: an event may appear following the triggering of an alarm. A condition can be made of several elementary conditions. It at least includes the elementary condition: presence of

the service trigger event.

- R= {r_1, r_2, ..., r_i, ..., r_p}, set of resources which may be necessary for the establishment of the services (if the services have visible actions). A resource can be made of several elementary resources. For example, it may be necessary to have a display screen, a printer or another peripheral device (possibly a sound device) in order to inform the user of the type of alarm message. The information contained in the fields of each resource relates particularly to its size, color(s) and all kinds of information contributing to the characteristics of the human-computer interface.
- P: Properties of the services. Each service results from the execution:
    1. either of a non visible action ac_N (an action which affects the agent itself): for example the service which deals with the displaying of a value of a process variable where the display service alone is involved in updating the value of the variable in question; we then speak about an internal property of the service.
    2. or of a visible action ac_V (an action which relates to another agent); for example actions of the human operator which consist in writing a message with a goal to send it to another person, or the change of a value of a process parameter; we then speak about an external property of the service.

A service can have two properties at the same time: in other words, following the appearance of an event, an agent can act on itself (such as the service which is in charge of the reactualization of a value of a variable) and on another agent at the same time (for instance in the case of exceeding a tolerated threshold of the value of the variable); in this case it will be necessary to generate and display an alarm message, which is the subject of another service.

An external action can relate to several agents (for instance, sending the same message to several agents); it will be regarded as a vector which includes the number of the action and the list of the identifiers of the agents concerned. If several screens of the human-machine interface are concerned with the same message, the agents of message acquisition, treatment and display each perform a different service.

3.4 Mathematical model

A mathematical formulation of the PN agent can be put forward, using (Moldt et Wienberg, 1997) as a starting point:

- An agent a_j is a set S_j of n services, S_j= {s_1, s_2, ..., s_i, ..., s_n}.
- For each service is associated an event e_i belonging to the set E of the events, E= {e_1, e_2, ..., e_i, …, e_n}.
- To each service, a condition c_i is associated, belonging to the set C of the conditions, C= {c_1, c_2, ..., c_i, ..., c_n}; a condition can be composed of several elementary conditions.
- If the service comprises a visible action, a resource r_i belonging to the set R of the resources is necessary, R= {r_1, r_2, ..., r_i, ..., r_p}.
- the set of the actions of the agent is composed of two subsets; AC_V is the set of visible actions and AC_N corresponds to the set of non visible actions.
  AC_V = {ac_{V1}, ac_{V2},…, ac_{VP}}, AC_N= {ac_{N1}, ac_{N2},…, ac_{Nq}} with p ≤ n and q ≤ n.

We have:

Card (E) = n, Card (C) = n, Card (R) = p, Card (AC_V) = p, Card (AC_N) = q.

A service results from a minimum of one action (which can be visible in the form of a

display, input coming from a keyboard, a click on the mouse, speech acquisition, …, or non visible, for instance when there are interactions between internal agents, such as the control agents) and a maximum of two actions (visible and non visible).
The number of all the services of the agent is defined by:
Nb_Services = Card (E) = n.
The number of all the actions of the agent is defined by:
Nb_Actions = Card (AC_V) + Card (AC_N)= p + q.
It is included in the margin: $n \leq Nb\_Actions \leq 2 \times n$.
We define the result of the service with a couple of actions (AC_{Vk}, AC_{Nt}), where the indices k and t take the zero value if the service does not contain a visible action or a non visible action. The number of the couples of actions is defined by:
Nb_Couples = Nb_Actions – Card (E)
We will order for example the couples of actions so that the couples which contain visible and non visible actions are the first, then the couples which contain only visible actions are in second place, while those which contain only non visible actions are in last place.
Then the services are defined by the following parameterized function (the parameter j being the identifier of the agent):

$S_j: E_j \times C_j \times R_j \longrightarrow AC_V \times AC_N.$

$(e_{i,j}; c_{i,j}; r_{k,j}) \longrightarrow (ac_{vk,j}; ac_{Nt,j})$. j = 1,…, number of agents
i = 1,...,n.
$0 \leq k \leq p$ such as if $i \leq p$ then k=i.
else k=0.
$0 \leq t \leq q$ such as if $i \leq Nb\_Couples$ then t=i
else
if i > p then t = i – p + Nb_Couples.
else t=0.

j: identifier of the agent
i: number of the event
AC_V: to express that it is a Visible action.
AC_N: to express that it is a Not Visible action.
k,t: number of the action (if the service does not result by a visible action, then k=0; if the service does not result by a non visible action, then t=0).
Then the specification of an agent a_j consists in the definition of the sets E_i , C_i, R_k and the mathematical specification of a task of an agent consists of the definition of a sequence of triplets (e_i, c_i, r_k) associated with their couples of actions ($ac_{vk,j;}$ $ac_{nt,j}$), by chronological order of appearance of the events.

4. Case study

The case study relates to an application of supervision of a complex process. It is supposed that a human operator (or group of human operators) is located in a control room: he or she must supervise it by the intermediary of an interactive system. The human operator intervenes in various normal situations as well as in abnormal ones. In the abnormal situations, it must intervene following the arrival of disturbances, or even anticipate them (Stanton, 1994; Moray, 1997). It is supposed that the application relates to the supervision of an urban transport network (such as tramways, buses...).
The architecture suggested for the interactive system consists of three modules: *Application,*

Control and *Presentation* (figure 5). Each module is composed of agents interacting between themselves and/or with the agents of other modules. Let us recall that an agent (set of n services) is defined by a quadruplet as follows:
Agent={E, C, R, P}.
It is the presentation module (composed of interactive agents) which we will model according to the following scenario of disturbance:

1. An event e_1 comes from a *dialogue controller* agent, following a disturbance in the application.
2. There is Ac_8 action of agent a_1 (a request for service of the agent a_1 to the agent a_2), following an analysis of the condition c_1 for the execution of the action a_1; this Ac_8 action in its turn becomes the event release e_5 for the agent a_2 (for example for the display of a message).
3. Agent a_2 carries out the Ac_3 action which consists in presenting the message to the user if the resource "Window" is available.
4. The event e_3 occurs; it represents the reaction of the user, following the display of the message. It consists of an input of a text with the keyboard or an acknowledgement of the message with the keyboard or the mouse.
5. The agent a_3 carries out the Ac_5 action following an event trigger e_3 which can be a return of an acknowledgement message or the validation of an action provided that c_3 is checked and that the display resource is available.

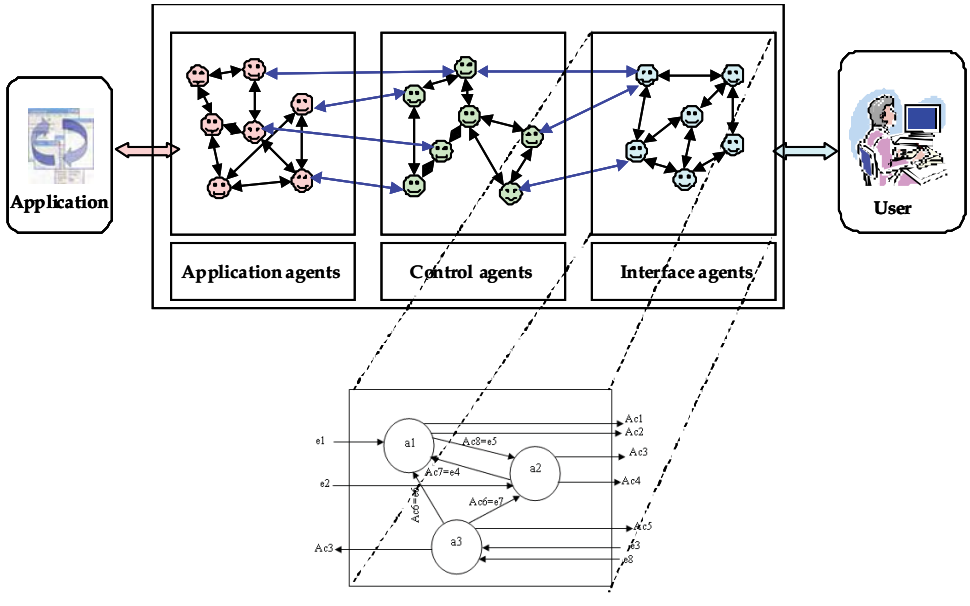


Fig. 5. Global view of the agent-based architecture, with arrival of a disturbance (Rehim, 2005).

While reformulating more formally, using (Moldt & Wienberg, 1997) as a source of inspiration, the presentation module can be defined by the following elements (figure 6):
A= {a1, a2, a3}: set of the presentation agents (in the example of figure 6, all three of them are

visible to the user).

E= {$e_{1,1;}$ $e_{1,2}$; $e_{1,3}$, $e_{2,3}$,; $e_{2,1}$; $e_{2,2}$; $e_{3,1}$; $e_{3,2}$}: set of events coming (1) from the agents of the *interface with the application* module (via the *dialogue controller* agents which send $e_{1,1}$ and $e_{1,2}$ events to the presentation module), such as alerts, dysfunctional events, incidents, and so on (2) from user actions such as: commands or confirmations ($e_{1,3;}$ $e_{2,3)}$, (3) from non visible actions which become events for other agents ($e_{2,1}$; $e_{2,2}$; $e_{3,1}$; $e_{3,2}$).

AC= { $ac_{v1,1;}$ $ac_{v2,1;}$ $ac_{v1,2;}$ $ac_{v2,2}$; $ac_{N1,1;}$ $ac_{N1,2;}$ $ac_{N1,3;}$ $ac_{N2,3;}$ $ac_{N3,3}$}: set of agent actions; these actions can be visible ($ac_{v1,1;}$ $ac_{v2,1;}$ $ac_{v1,2;}$ $ac_{v2,2;}$ $ac_{v1,3}$), such as the display of information on a screen, or non visible ($ac_{N1,1;}$ $ac_{N1,2;}$ $ac_{N1,3;}$ $ac_{N2,3;}$ $ac_{N3,3}$), such as a request for service to other agents.

R= {$r_{1,1}$; $r_{2,1;}$ $r_{1,2;}$ $r_{2,2;}$ $r_{1,3}$ }: set of resources (such as keyboard, mouse, windows…) necessary for the visible actions of the agents, in order to carry out their actions ($r_{1,1}$ is related to $ac_{v1,1}$, $r_{2,1}$ is related to $ac_{v2,1}$, and so on).

C= {$c_{v1,1;}$ $c_{v2,1;}$ $c_{v1,2;}$ $c_{v2,2}$; $c_{N1,1;}$ $c_{N1,2;}$ $c_{N1,3;}$ $c_{N2,3;}$ $c_{N3,3}$}: set of conditions related to the execution of the visible or non visible actions ($c_{v1,1}$ is related to $ac_{v1,1}$, $c_{v2,1}$ is related to $ac_{v2,1}$, and so on).

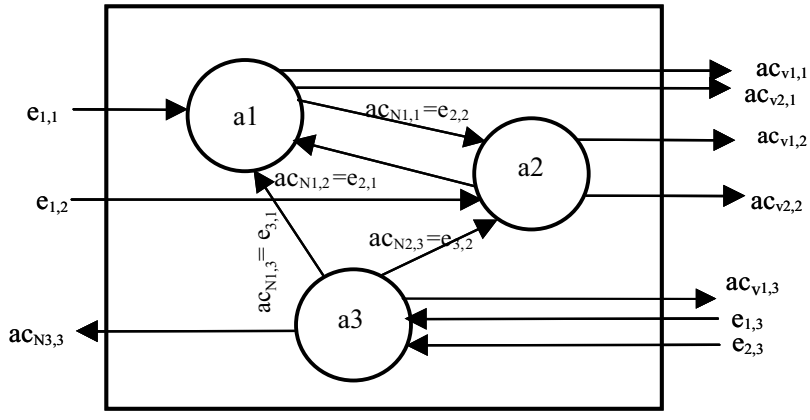


Fig. 6. Reformulated example related to the disturbance (*presentation* module)

For a better representation of the example visible in figure 6, one associates an agent identifier i,j with all the information present in the example; i is the event, action or necessary resource number; j is the agent number (from 1 to N). The possible couples constituted with visible ($ac_{vi,j}$) and non visible ($ac_{Ni,j}$) actions are obtained following the appearance of an $e_{i,j}$ event, under the conditions that $c_{i,j}$ is true and the $r_{i,j}$ resource is available. The modeling of services of each of the three agents forming the presentation module can be expressed as follows, with $r_{0,j}$: resource not necessary for the execution of the non visible action ($ac_{N0,j}$) by an agent j, and with $ac_{v0,j}$ and $a_{N0,j}$: visible and non visible actions which are not performed by the agent j.

Agent1:

E1 = {$e_{1,1;}$ $e_{2,1}$; $e_{3,1}$}

Ac_{v1} = {$ac_{v1,1}$; $ac_{v2,1}$}, Ac_{N1} = {$ac_{N1,1}$},

R = {$r_{1,1}$; $r_{2,1}$},

C = {$c_{v1,1}$; $c_{v2,1}$; $c_{N1,1}$}.

Card (E_1) = 3, Card (Ac_{v1}) = 2, Card (Ac_{N1}) = 1.
Nb_Actions = Card (Ac_{v1}) + Card (Ac_{N1}) = 2+1=3.
Nb_Couples = Nb_Actions – Nb_Services = 3 – 3 = 0.
S1: $E_1 \times C_1 \times R_1 \longrightarrow Ac_{v1} \times Ac_{N1}$
$(e_{1,1}; c_{v1,1}; r_{1,1}) \longrightarrow (ac_{v1,1}; ac_{N1,1})$
$(e_{2,1}; c_{v2,1}; r_{2,1}) \longrightarrow (ac_{v2,1}; ac_{N1,1})$
$(e_{3,1}; c_{N1,1}; r_{0,1}) \longrightarrow (ac_{v0,1}; ac_{N1,1})$

Agent2:
$E_2 = \{e_{1,2}; e_{2,2;} e_{3,2}\}$,
$Ac_{v2} = \{ac_{v1,2}; ac_{v2,2}\}$, $Ac_{N2} = \{ac_{N1,2}\}$,
$R_2 = \{r_{1,2}; r_{2,2}\}$,
$C_2 = \{c_{v1,2}; c_{v2,2}; c_{N1,2}\}$.

Card (E_2) = 3, Card (Ac_{v2}) = 2, Card (Ac_{N2}) = 1.
Nb_Actions = Card (Ac_{v2}) + Card (Ac_{N2}) = 2+1=3.
Nb_Couples = Nb_Actions – Nb_Services = 3 – 3 = 0.
S_2: $E_2 \times C_2 \times R_2 \longrightarrow Ac_{v2} \times Ac_{N2}$
$(e_{1,2}; c_{v1,2}; r_{1,2}) \longrightarrow (ac_{v1,2}; ac_{N1,2})$
$(e_{2,2}; c_{v2,2}; r_{2,2}) \longrightarrow (ac_{v2,2}; ac_{N1,2})$
$(e_{3,2}; c_{N1,2}; r_{0,2}) \longrightarrow (ac_{v0,2}; ac_{N1,2})$

Agent3:
$E_3 = \{e_{1,3}; e_{2,3}\}$,
$Ac_{v3} = \{ac_{v1,3}\}$, $Ac_{N3} = \{ac_{N1,3}; ac_{N2,3;} a_{N3,3}\}$,
$R_3 = \{r_{1,3}\}$,
$C_3 = \{c_{v1,3}; c_{N1,3}; c_{N2,3;} c_{N3,3}\}$.

Card (E_3) = 2, Card (Ac_{v3}) = 1, Card (Ac_{N3}) = 3.
Nb_Actions = Card (Ac_{v3}) + Card (Ac_{N3}) = 1+3=4.
Nb_Couples = Nb_Actions – Nb_Services = 4 – 4 = 0.
S3: $E_3 \times C_3 \times R_3 \longrightarrow Ac_{v3} \times Ac_{N3}$
$(e_{1,3}; c_{v1,3}; r_{1,3}) \longrightarrow (ac_{v1,3;} ac_{N1,3})$
$(e_{2,3}; c_{N2,3}; r_{0,3}) \longrightarrow (ac_{v0,3}; ac_{N2,3})$
$(e_{3,3}; c_{N3,3}; r_{0,3}) \longrightarrow (ac_{v0,3}; ac_{N3,3})$

From the mathematical modeling above, which made it possible to formulate the interactions of the agents in figure 6, each agent is modeled with the determination of its inputs and outputs in terms of condition, action, event and resource necessary to achieve the service. In figure 7, we present a model of interaction between the agents of the *presentation* module of the human-machine interface.

Figure 8 shows an example of release of the service $s_{7,1}$ "Change_Delay_Threshold_Vehicle" of the agent called *Traffic_State* belonging to the presentation module of the human-machine interface. Following the appearance of the event $e_{7,1}$ "Delay_Threshold_Vehicle" the interface agent *Vehicle* checks the corresponding condition $c_{7,1}$ "Appearance of the event and delay > 5". If the condition is verified, the service is started; the visible action $ac_{v7,1}$ is activated. The activation of this action exploits the resource $r_{7,1}$ "Dialog_Box" and reveals an alarm message bound for the human operator.

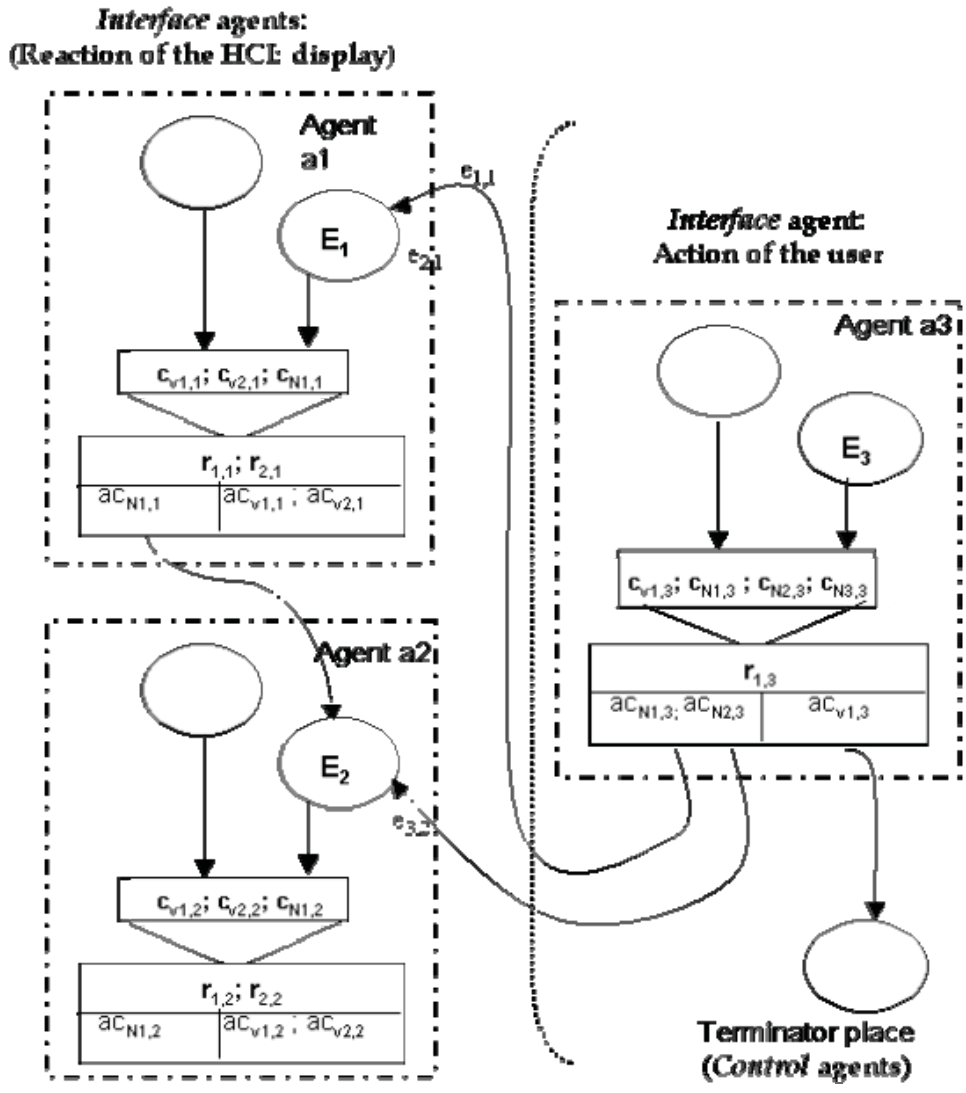


Fig. 7. Modeling of the example related to the disturbance (Rehim, 2005)

We notice, with the example presented above (figure 8), that there is no non visible action ($ac_{N0,1}$) of agent 1 in transition T1, i.e. that there is no interaction with other internal agents; but on the other hand, thanks to the visible action $ac_{v7,1}$, agent 1 acts on an external agent which is the window n°2 belonging to the presentation module of the interactive system.

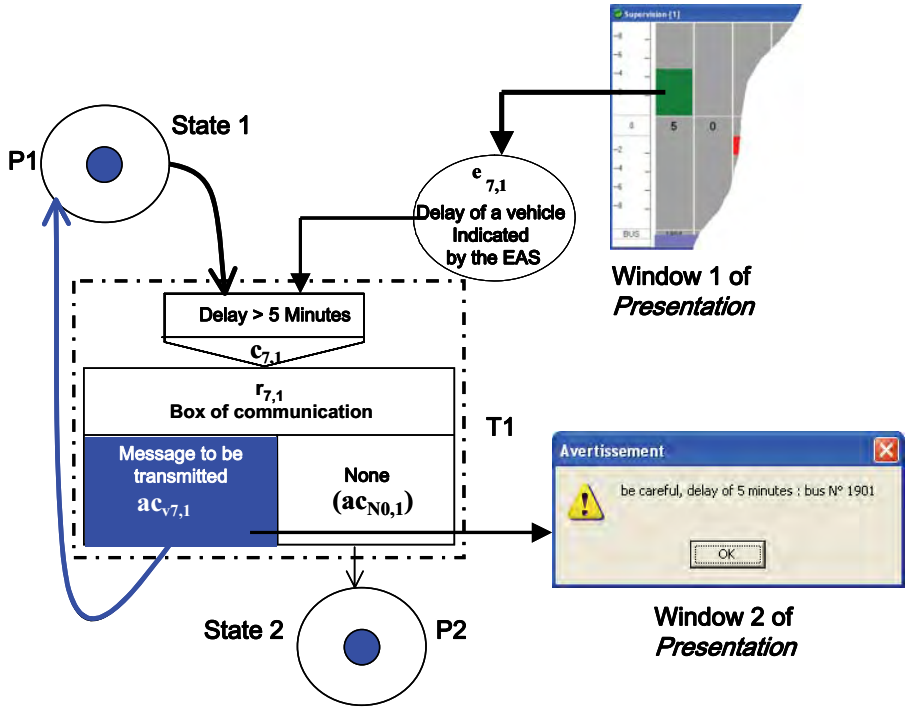


Fig. 8. Example of release of a service of the *Traffic_State* agent (Trabelsi, 2006)

4. Conclusion

Through their formal aspect and their capacity to model the dynamics of systems, Petri Nets have been bringing complementary and significant contributions in the human-computer interaction domain since the end of the Eighties. In this chapter, we have explained their utility for the modeling of agents which make up interactive systems with an architecture containing agents. A scenario made it possible to illustrate the approach proposed.

The PN used represent a promising tool for the modeling of such interactive systems. Their originality and their power reside in (1) their capacity to visualize the behavior of each agent (external or internal actions), as well as (2) their capacity of abstraction which makes it possible to keep the same information without any visual complexity, and (3) their formal aspect also enables them to be potentially efficient at the time of the evaluation and validation phase (which is not dealt with in this article).

There are several perspectives with this work. We are currently studying and developing an assistance tool for the evaluation of interactive systems. This tool makes it possible to connect to each interface agent, evaluation agents intended to analyze their behavior at the time of situations of use. The PN must make it possible to

reconstitute the human activities performed (Trabelsi, 2006; Tran et al., 2007). A second perspective relates to generalization with the agents of the two other modules: (1) interface with the application, (2) dialogue controller. Another perspective relates to the evaluation of the approach suggested in various application domains (for instance design and evaluation web sites).

5. Acknowledgements

The authors thank the FEDER, the GRRT and the Nord-Pas de Calais region for their financial support (SART, EUCUE and MIAOU projects). They also thank André Péninou, Hacène Maoudji, Aïssam Rehim and Abdelwaheb Trabelsi for their contribution to various modeling aspects described in this chapter.

6. References

Abed, M. (1990). Contribution à la modélisation de la tâche par des outils de spécification exploitant les mouvements oculaires : application à la conception et à l'évaluation des interfaces homme-machine. Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis, septembre.

Abed, M, Bernard, J.M, Angué, J.C (1992). Method for comparing task model and activity model. Proceedings 11th European annual conference Human Decision Making and Manual Control, Valenciennes, France.

Tabary, D., Abed, M. (1998). TOOD: an object-oriented methodology for describing user task in interface design and specification - An application to air traffic control. La Lettre de l'Intelligence Artificielle, 134, pp. 107-114.

Abed, M. (2001). Méthodes et modèles formels et semi-formels pour la conception et l'évaluation des systèmes homme-machine. Habilitation à diriger des recherches, Université de Valenciennes et du Hainaut-Cambrésis, 02 mai 2001.

Benaïssa, M.L., Ezzedine, H., Angué, J.C. (1993). An interface Specification Method for industrial processes. XII European annual conference on human decision making and manual control, Kassel, Germany, juin.

Bernonville, S., Leroy, N., Kolski, C., Beuscart-Zéphir, M. (2006). Explicit combination between Petri Nets and ergonomic criteria: basic principles of the ErgoPNets method. Proceedings of the 25th Edition of EAM'06, European Annual Conference on Human Decision-Making and Manual Control (September 27-29, 2006, Valenciennes, France), PUV.

Coutaz, J. (1987). PAC, an Object-Oriented Model for Dialog Design. In: Bullinger, Hans-Jorg, Shackel, Brian (ed.): INTERACT 87 - 2nd IFIP International Conference on Human-Computer Interaction. September 1-4, Stuttgart, Germany. p.431-436.

David, R. & Alla, H. (2004). Discrete, continuous, and hybrid Petri Nets. 1er ed. Springer Verlag, 2004, XXII, 524 p. Hardcover ISBN 3-540-22480-7

Ezzedine, H., Kolski, C. (2005). Modelling of cognitive activity during normal and abnormal situations using Object Petri Nets, application to a supervision system. Cognitive, Technology and Work, 7, pp. 167-181.

Ezzedine, H., Trabelsi, A., Kolski, C. (2006). Modelling of agent oriented interaction using Petri Nets, application to HMI design for transport system supervision. P. Borne, E.

Craye, N. Dangourmeau (Ed.), *CESA2003 IMACS Multiconference Computational Engineering in Systems Applications (Lille, France, July 9-11, 2003)*, Ecole Centrale Lille, Villeneuve D'Ascq, pp. 1-8, janvier, ISBN 2-9512309-5-8.

Ezzedine, H., Trabelsi, A., Kolski, C. (2006). Modelling of an interactive system with an agent-based architecture using Petri nets, application of the method to the supervision of a transport system. *Mathematics and Computers in Simulation*, 70, pp. 358-376.

Ezzedine, H., Trabelsi, A. (2005). From the design to the evaluation of an agent-based human-machine interface. Application to supervision for urban transport system. P. Borne, M. Benrejeb, N. Dangoumeau, L. Lorimier (Ed.), *IMACS World Congress "Scientific Computation, Applied Mathematics and Simulation" (July 11-15, Paris)*, ECL, pp. 717-725, juillet, ISBN 2-915913-02-1.

Ezzedine, H., Maoudji, H. & Péninou A. (2001). Towards agent oriented specification of Human-Machine Interface : Application to the transport systems. 8th IFAC Symposium on Analysis, Design, and Evaluation of Human-Machine Systems (IFAC-HMS 2001), pp. 421-426, Kassel, Germany, 18-20 September.

Foley, J.D & Van Dam, A. (1982). Fundamentals of Interactive Computer Graphics, Addison-Wesley (IBM Systems Programming Series), Reading, MA.

Goldberg, A. (1980). Smalltalk-80, the interactive programming environnement. Addision-Wesley.

Gomes, L., Barros, J.P., Coasta, A. (2001). Man-machine interface for real-time telecontrol based on Petri nets specification. In T. Bahill, F.Y. Wand (Eds.), IEEE SMC 2001 Conference Proceedings (e-Systems, e-Man and e-Cybernetics), Arizona, USA: IEEE Press, pp. 1565-1570.

Gracanin, D., Srinivasan, P. & Valavanis, K.P. (1994). Parametized Petri nets and their applications to planning and coordination in intelligent systems. IEEE Transactions on Systems, Man and Cybernetics, Vol. 24, pp. 1483-1497.

Guittet, L. (1995). Contribution à l'Ingénierie des IHM - Théorie des Interacteurs et Architecture H4 dans le système NODAOO, Thèse de l'Université de Poitiers, 1995.

Jensen, K. (1980). Coloured Petri Nets and Invarient Method. Daimi PB 104, Aarhus University.

Jensen, K. (1996). Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use. 2nd edition, vol n°2, Springer-Verlag.

Kontogiannis, T. (2003). A Petri Net-based approach for ergonomic task analysis and modeling with emphasis on adaptation to system changes. Safety Science, vol. 41 n°10, pp. 803-835.

Kaddouri, S.A., Ezzedine, H., Angué, J.C. (1995). Task modelling using object Petri Nets. In Anzaï Y., Ogawa K., Mori H. (Eds.), Symbiosis of Human and Artefact, HCI International'95: 6th International, Tokyo, Japan. (pp. 988-994). Amsterdam: Elsevier.

Mahfoudhi, A., Abed, M., Angué, J.C. (1995). An Object Oriented Methodology for Man-Machine systems analysis and design. Anzai Y., Ogawa K., Mori H. (Ed.), Symbiosis of Human and Artefact, HCI International'95: 6th International, Tokyo, Japan, Elsevier, Amsterdam, pp. 965-970, janvier.

Maoudji, H., Ezzedine, H. & Péninou A. (2001). Agents oriented specification of interactive systems. In M.J. Smith, G. Salvendy, D. Harris, R. Koubek (Ed.), Usability

evaluation and Interface design: Cognitive Engineering, Intelligent Agents and Virtual Reality, volume 1. (pp. 71-75). London : Lawrence Erlbaum Associate Publishers.

Maoudji, H., Ezzedine, H., Péninou, A. & Kolski, C. (2000). Amélioration de la qualité des correspondances dans les réseaux de transports urbains. Rapport d'étude à mi-parcours du projet coopératif GRRT, Juillet.

Moldt, M., Wienberg, F. (1997). Multi-Agent-Systems based on Coloured Petri Nets. In Proceedings of the 18th International Conference on Application and Theory of Petri Nets, Toulouse.

Moray, N. (1997). Human factors in process control. In Handbook of human factors and ergonomics, G. Salvendy (Ed.), John Wiley & Sons, INC., pp. 1944-1971.

Moussa, F., Riahi, M., Kolski, C., Moalla, M. (2002). Interpreted Petri Nets used for Human-Machine Dialogue Specification in Process Control : principles and application to the Ergo-Conceptor+ tool. Integrated Computer-Aided Engineering, 9, pp. 87-98.

Moussa, F., Kolski, C., Riahi, M. (2006). Analyse des dysfonctionnements des systèmes complexes en amont de la conception des IHM : apports, difficultés, et étude de cas. Revue d'Interaction Homme Machine (RIHM), 7, pp. 79-111.

Navarre, D., Palanque, P., Bastide, R. (2003). A Tool-Supported Design Framework for Safety Critical Interactive Systems, Interacting with computers, 15 (3), pp. 309-328.

Nigay, L., Coutaz, J. (1997). Software architecture modelling: Bridging Two Worlds using Ergonomics and Software Properties. In Formal Methods in Human-Computer Interaction, P. Palanque & F. Paterno (Eds.), Springer-Verlag: London Publ., ISBN 3-540-76158-6, 1997, pp. 49-73.

Ouadou, K. (1994). AMF : Un modèle d'architecture multi-agents multi-facettes pour Interfaces Homme-Machine et les outils associés. Thèse de l'Ecole Centrale de Lyon. 1994.

Palanque, P. & Bastide, R. (1995). Design, specification and of interactive systems. Springer Verlag 1995, ISBN 3-211-82739-0. 370 pages.

Palanque, P., Bastide, R. (1990). Petri nets with objects for specification, design and validation of user-driven interfaces. In proceedings of the third IFIP conference on Human-Computer Interaction, Interact'90, Cambridge,UK, 27-31 August.

Palanque, P. (1992). Modélisation par objets coopératifs interactifs d'interfaces homme-machines dirigées par l'utilisateur. Ph.D. Thesis, University of Toulouse 1, France.

Palanque, P., Bastide, R. (1997). Synergistic modelling of tasks, system and users using formal specification techniques. Interacting With Computers, 9 (12), pp. 129-153.

Palanque, P., Bastide, R., Sengès, V. (1995). Task model–system model: towards an unifying formalism. In proceedings of the HCI international (EHCI'95), Chapman & Hall, pp. 189-212.

Palanque, P., Farenc, C., Bastide, R. (1999). Embedding Ergonomic Rules As Generic Requirements in a formal Development Process of Interactive Software. In Proceeding Interact'99, Sasse A., Jonhson C. (Eds), IOS Press, pp. 408-416.

Pfaff, (1985). User interface management system. Springer-Verlag.

Rehim, A. (2005). Etude des outils exploitant des réseaux de Petri agent pour l'évaluation des systèmes interactifs. Mémoire de Master recherche.

Sibertin-Blanc, C. (1985). High-level Petri nets with Data Structure. Proceedings 6th EWPNA, June, Espoo, Finland, 1985.

Stanton, N. (1994). Human factors in alarm design. Taylor & Francis Ltd, London.

Tabary, D., Abed, M. (2002). A software Environment Task Object Oriented Design (ETOOD). Journal of Systems and Software, 60, pp.129-141.

Tarpin-Bernard, F. & David, B. (1999). AMF : un modèle d'architecture multi-agents multi-facettes Techniques et Sciences Informatiques. Hermès. Paris Vol. 18 No. 5. pp. 555-586. Mai . Thèse de doctorat, Université Joseph Fourier Grenoble 1, Mars.

Trabelsi, A. (2006). Contribution à l'évaluation des systèmes interactifs orientés agents. Application à un poste de supervision de transport urbain. Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis, 25 septembre.

Trabelsi, A., Ezzedine, H. & Kolski, C. (2006). Un mouchard électronique orienté agent pour l'évaluation de systèmes interactifs de supervision. CIFA2006, Bordeaux, France, 30-31 Mai et 1 juin. Université de Valenciennes et du Hainaut-Cambrésis, juillet.

Tran, C.D., Ezzedine, H. & Kolski, C. (2007). Towards a generic and configurable model of an electronic informer to assist the evaluation of agent-based interactive systems.

ICEIS'2007, 9th International Conference on Entreprise Information Systems. 12-16 June, Funchal, Madeira- Portugal

Williem, R., Biljon, V. (1988). Extending Petri Nets for specifying Man-Machine dialogues. International Journal of Man-Machine Studies, vol. 28, pp. 437-45.

Winckler, M., Barboni, E., Palanque, P., Farenc., C. (2006). What Kind of Verification of Formal Navigation Modelling for Reliable and Usable Web Applications? 1st Int. Workshop on Automated Specification and Verification of Web Sites, Valencia, Spain. Electronic Notes Theoretical Computer Science, 157(2), pp. 207-211.

8

On the Use of Queueing Petri Nets for Modeling and Performance Analysis of Distributed Systems

Samuel Kounev and Alejandro Buchmann
Technische Universität Darmstadt
Germany

1. Introduction

Predictive performance models are used increasingly throughout the phases of the software engineering lifecycle of distributed systems. However, as systems grow in size and complexity, building models that accurately capture the different aspects of their behavior becomes a more and more challenging task. The challenge stems from the limited model expressiveness on the one hand and the limited scalability of model analysis techniques on the other. This chapter presents a novel methodology for modeling and performance analysis of distributed systems [Kounev, 2006]. The methodology is based on queueing Petri nets (QPNs) which provide greater modeling power and expressiveness than conventional modeling paradigms such as queueing networks and generalized stochastic Petri nets. Using QPNs, one can integrate both hardware and software aspects of system behavior into the same model. In addition to hardware contention and scheduling strategies, QPNs make it easy to model software contention, simultaneous resource possession, synchronization, blocking and asynchronous processing. These aspects have significant impact on the performance of modern distributed systems.

To avoid the problem of state space explosion, our methodology uses discrete event simulation for model analysis. We propose an efficient and reliable method for simulation of QPNs [Kounev & Buchmann, 2006]. As a validation of our approach, we present a case study of a real-world distributed system, showing how our methodology is applied in a step-by-step fashion to evaluate the system performance and scalability. The system studied is a deployment of the industry-standard SPECjAppServer2004 benchmark. A detailed model of the system and its workload is built and used to predict the system performance for several deployment configurations and workload scenarios of interest. Taking advantage of the expressive power of QPNs, our approach makes it possible to model systems at a higher degree of accuracy providing a number of important benefits.

The rest of this chapter is organized as follows. In Section 2, we give a brief introduction to QPNs. Following this, in Section 3, we present a method for quantitative analysis of QPNs based on discrete event simulation. The latter enables us to analyze QPN models of realistic size and complexity. In Section 4, we present our performance modeling methodology for distributed systems. The methodology is introduced in a step-by-step

fashion by considering a case study in which QPNs are used to model a real-life system and analyze its performance and scalability. After the case study, some concluding remarks are presented and the chapter is wrapped up in Section 5.

2. Queueing Petri nets

Queueing Petri Nets (QPNs) can be seen as a combination of a number of different extensions to conventional Petri Nets (PNs) along several different dimensions. In this section, we include some basic definitions and briefly discuss how QPNs have evolved. A deeper and more detailed treatment of the subject can be found in [Bause, 1993].

2.1 Evolution of queueing Petri nets

An ordinary *Petri net* (also called *place-transition net*) is a bipartite directed graph composed of places, drawn as circles, and transitions, drawn as bars. A formal definition is given below [Bause and Kritzinger, 2002]:

Definition 1 *An ordinary Petri Net (PN) is a 5-tuple* $PN = (P, T, I^-, I^+, M_0)$ *where:*

1. $P = \{p_1, p_2, ..., p_n\}$ *is a finite and non-empty set of places,*
2. $T = \{t_1, t_2, ..., t_m\}$ *is a finite and non-empty set of transitions,* $P \cap T = \emptyset$,
3. $I^-, I^+ : P \times T \rightarrow \mathbb{N}_0$ *are called backward and forward incidence functions, respectively,*
4. $M_0 : P \rightarrow \mathbb{N}_0$ *is called initial marking.*

The incidence functions I^- and I^+ specify the interconnections between places and transitions. If $I^-(p,t) > 0$, an arc leads from place *p* to transition *t* and place *p* is called an *input place* of the transition. If $I^+(p,t) > 0$, an arc leads from transition *t* to place *p* and place *p* is called an *output place* of the transition. The incidence functions assign natural numbers to arcs, which we call *weights* of the arcs. When each input place of transition *t* contains at least as many tokens as the weight of the arc connecting it to *t,* the transition is said to be *enabled.* An enabled transition *may fire,* in which case it destroys tokens from its input places and creates tokens in its output places. The amounts of tokens destroyed and created are specified by the arc weights. The initial arrangement of tokens in the net (called *marking)* is given by the function M_0, which specifies how many tokens are contained in each place.

Different extensions to ordinary PNs have been developed in order to increase the modeling convenience and/or the modeling power. *Colored* PNs (CPNs) introduced by K. Jensen are one such extension [Jensen, 1981]. The latter allow a type (color) to be attached to a token. A color function C assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different *modes* (transition colors). The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis. A formal definition of a CPN follows [Bause & Kritzinger, 2002]:

Definition 2 *A Colored PN (CPN) is a 6-tuple* $CPN = (P, T, C, I^-, I^+, M_0)$ *where:*

1. $P = \{p_1, p_2, ..., p_n\}$ is a finite and non-empty set of places,
2. $T = \{t_1, t_2, ..., t_m\}$ *is a finite and non-empty set of* transitions, $P \cap T = \emptyset$,
3. *C is a color function that assigns a finite and non-empty set of colors to each place and a finite and non-empty set of modes to each transition.*
4. I^- *and* I^+ *are the backward and forward incidence functions defined on* $P \times T$, *such that*

$I^-(p,t), I^+(p,t) \in [C(t) \to C(p)_{MS}], \forall (p,t) \in P \times T$[1]

5. M_0 *is a function defined on P describing the initial marking such that* $M_0(p) \in C(p)_{MS}$

Other extensions to ordinary PNs allow temporal (timing) aspects to be integrated into the net description [Bause & Kritzinger, 2002]. In particular, *Stochastic* PNs (SPNs) attach an exponentially distributed *firing delay* to each transition, which specifies the time the transition waits after being enabled before it fires. *Generalized Stochastic* PNs (GSPNs) allow two types of transitions to be used: immediate and timed. Once enabled, immediate transitions fire in zero time. If several immediate transitions are enabled at the same time, the next transition to fire is chosen based *on firing weights* (probabilities) assigned to the transitions. Timed transitions fire after a random exponentially distributed firing delay as in the case of SPNs. The firing of immediate transitions always has priority over that of timed transitions. A formal definition of a GSPN follows [Bause & Kritzinger, 2002]:

Definition 3 *A Generalized SPN (GSPN) is a 4-tuple* $GSPN = (PN, T_1, T_2, W)$ *where:*

1. $PN = (P, T, I^-, I^+, M_0)$ *is the underlying ordinary PN,*
2. $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$,
3. $T_2 \subset T$ *is the set of immediate transitions,* $T_1 \cap T_2 = \emptyset, T_1 \cup T_2 = T$,
4. $W = (w_1, ..., w_{|T|})$ *is an array whose entry* $w_i \in \mathbb{R}^+$ *is a rate of a negative exponential distribution specifying the firing delay, if* $t_i \in T_1$ *or is a firing weight specifying the relative firing frequency, if* $t_i \in T_2$.

Combining CPNs and GSPNs leads to Colored GSPNs (CGSPNs) [Bause & Kritzinger, 2002]:

Definition 4 *A Colored GSPN (CGSPN) is a 4-tuple* $CGSPN = (CPN, T_1, T_2, W)$ *where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying CPN,*
2. $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$,
3. $T_2 \subset T$ *is the set of immediate transitions,* $T_1 \cap T_2 = \emptyset, T_1 \cup T_2 = T$,
4. $W = (w_1, ..., w_{|T|})$ *is an array with* $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ *such that* $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}^+$ *is a rate of a negative exponential distribution specifying the firing delay due to color c, if* $t_i \in T_1$ *or is a firing weight specifying the relative firing frequency due to c, if* $t_i \in T_2$.

While CGSPNs have proven to be a very powerful modeling formalism, they do not provide any means for direct representation of queueing disciplines. The attempts to eliminate this disadvantage have led to the emergence of *Queueing PNs (QPNs).* The main idea behind the QPN modeling paradigm was to add queueing and timing aspects to the places of CGSPNs. This is done by allowing queues (service stations) to be integrated into places of CGSPNs. A place of a CGSPN that has an integrated queue is called a *queueing place* and consists of two components, the *queue* and a *depository* for tokens which have completed their service at the queue. This is depicted in Figure 1.

The behavior of the net is as follows: tokens, when fired into a queueing place by any of its input transitions, are inserted into the queue according to the queue's scheduling strategy. Tokens in the queue are not available for output transitions of the place. After completion of its service, a token is immediately moved to the depository, where it becomes available for output transitions of the place. This type of queueing place is called timed queueing place. In addition to timed queueing places, QPNs also introduce *immediate* queueing places, which allow pure scheduling aspects to be described. Tokens in immediate queueing places can be viewed as being served immediately. Scheduling in

[1] The subscript MS denotes multisets. $C(p)_{ms}$ denotes the set of all finite multisets of $C(p)$.

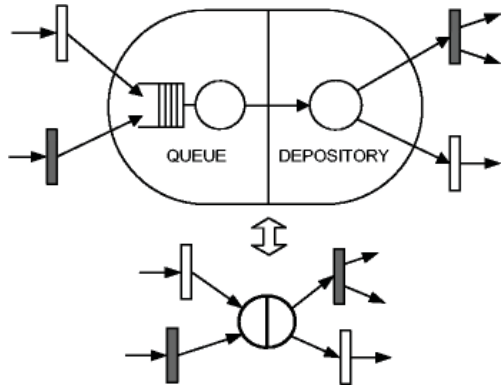


Fig. 1. A queueing place and its shorthand notation.

such places has priority over scheduling/service in timed queueing places and firing of timed transitions. The rest of the net behaves like a normal CGSPN. An enabled timed transition fires after an exponentially distributed delay according to a race policy. Enabled immediate transitions fire according to relative firing frequencies and their firing has priority over that of timed transitions. A formal definition of a QPN follows:

Definition 5 *A Queueing PN (QPN) is an 8-tuple* $QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ *where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying Colored* PN
2. $Q = (Q_1, Q_2, (q_1, ..., q_{|P|}))$ *where*
   - $\tilde{Q_1} \subseteq P$ *is the set of timed queueing places,*
   - $\tilde{Q_2} \subseteq P$ *is the set of immediate queueing places,* $\tilde{Q_1} \cap \tilde{Q_2} = \emptyset$ *and*
   - *qi denotes the description of a queue*[2] *taking all colors of C(pi) into consideration, if pi is*
   - a queueing place or equals the keyword 'null', if pi is an ordinary place.
3. $W = (\tilde{W_1}, \tilde{W_2}, (w_1, ..., w_{|T|}))$ *where*
   - $\tilde{W_1} \subseteq T$ *is the set of timed transitions,*
   - $\tilde{W_2} \subseteq T$ *is the set of immediate transitions,* $\tilde{W_1} \cap \tilde{W_2} = \emptyset$, $\tilde{W_1} \cup \tilde{W_2} = T$ *and*
   - $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ *such that* $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}^+$ *is interpreted as a rate of a negative exponential distribution specifying the firing delay due to color c, if* $t_i \in \tilde{W_1}$ or *a firing weight specifying the relative firing frequency due to color c, if* $t_i \in \tilde{W_2}$.

Example 1 (QPN) *Figure 2 shows an example of a QPN model of a central server system with memory constraints based on [Bause and Kritzinger, 2002]. Place* p_2 *represents several terminals, where users start jobs (modeled with tokens of color 'o') after a certain thinking time. These jobs request service at the CPU (represented by a G/C/l/PS queue, where C stands for Coxian distribution) and two disk subsystems (represented by G/C/1/FCFS queues). To enter the system each job has to allocate a certain amount of memory. The amount of memory needed by each job is*

[2] In the most general definition of QPNs, queues are defined in a very generic way allowing the specification of arbitrarily complex scheduling strategies taking into account the state of both the queue and the depository of the queueing place [Bause, 1993]. For the purposes of this chapter, it is enough to use conventional queues as defined in queueing network theory.

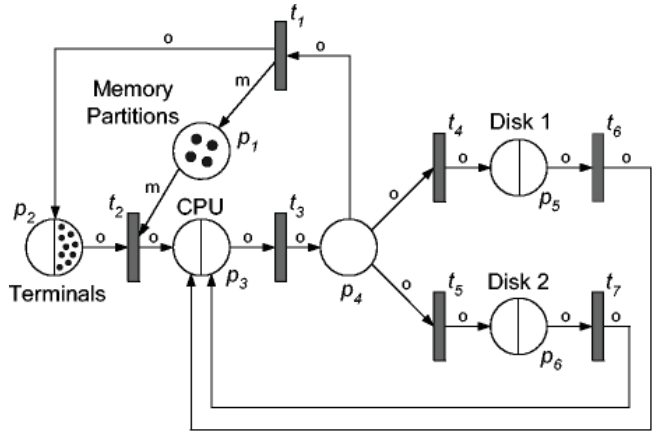


Fig. 2. A QPN model of a central server with memory constraints (reprinted from [Bause & Kritzinger, 2002]).

assumed to be the same, which is represented by a token of color 'm' on place p_1. Note that, for readability, token cardinalities have been omitted from the arc weights in Figure 2, i.e., symbol o stands for ***1'o*** *and symbol* ***m*** *for* ***1'm****. According to Definition 5, we have the following:*
$QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ *where*
$CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying Colored PN as depicted in Figure 2,*
$Q = (\tilde{Q}_1, \tilde{Q}_2, (null, G/C/\infty/IS, G/C/1/PS, null, G/C/1/FCFS, G/C/1/FCFS))$,
$\tilde{Q}_1 = \{p_2, p_3, p_5, p_6\}, \tilde{Q}_2 = \emptyset$,
$W = (\tilde{W}_1, \tilde{W}_2, (w_1, ..., w_{|T|}))$, *where* $\tilde{W}_1 = \emptyset, \tilde{W}_2 = T$ *and* $\forall c \in C(t_i) : w_i(c) := 1$, *so that all transition firings are equally likely.*

2.2 Hierarchical queueing Petri nets

A major hurdle to the practical application of QPNs is the so-called *largeness problem* or *state-space explosion problem:* as one increases the number of queues and tokens in a QPN, the size of the model's state space grows exponentially and quickly exceeds the capacity of today's computers. This imposes a limit on the size and complexity of the models that are analytically tractable. An attempt to alleviate this problem was the introduction of *Hierarchically-Combined QPNs (HQPNs)* [Bause *et al.*, 1994]. The main idea is to allow hierarchical model specification and then exploit the hierarchical structure for efficient numerical analysis. This type of analysis is termed *structured analysis* and it allows models to be solved that are about an order of magnitude larger than those analyzable with conventional techniques.

HQPNs are a natural generalization of the original QPN formalism. In HQPNs, a queueing place may contain a whole QPN instead of a single queue. Such a place is called a *subnet place* and is depicted in Figure 3. A subnet place might contain an ordinary QPN or again a HQPN allowing multiple levels of nesting. For simplicity, we restrict ourselves to two-level hierarchies. We use the term *High-Level QPN (HLQPN)* to refer to the upper level of the HQPN and the term *Low-Level QPN (LLQPN)* to refer to a subnet of the HLQPN. Every subnet of a HQPN has a dedicated input and output place, which are ordinary places of a CPN. Tokens being inserted into a subnet place after a transition firing are added to the input place of the corresponding HQPN subnet. The semantics of the output

place of a subnet place is similar to the semantics of the depository of a queueing place: tokens in the output place are available for output transitions of the subnet place. Tokens contained in all other places of the HQPN subnet are not available for output transitions of the subnet place. Every HQPN subnet also contains *actual – population* place used to keep track of the total number of tokens fired into the subnet place.

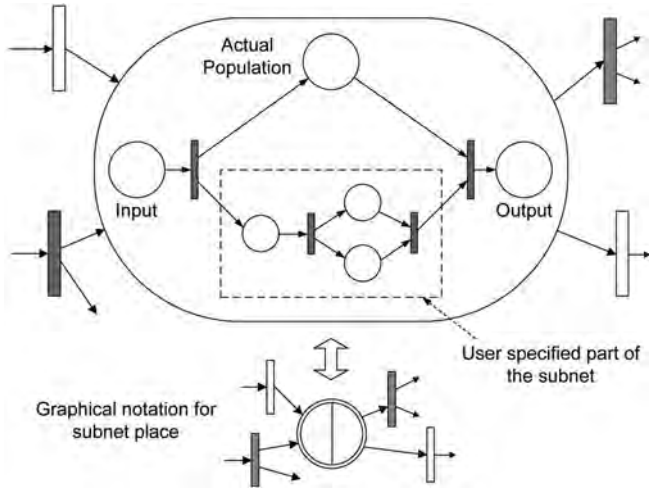


Fig. 3. A subnet place and its shorthand notation.

3. Quantitative analysis of queueing Petri nets

In [Kounev & Buchmann, 2003], we showed that QPNs lend themselves very well to modeling distributed e-business applications with software contention and demonstrated how this can be exploited for performance prediction in the capacity planning process. However, we also showed that modeling a realistic e-business application using QPNs often leads to a model that is way too large to be analytically tractable. While, HQPNs and structured analysis techniques alleviate this problem, they do not eliminate it. This is the reason why QPNs have hardly been exploited in the past 15 years and very few, if any, practical applications have been reported. The problem is that, until recently, available tools and solution techniques for QPN models were all based on Markov chain analysis, which suffers the well known *state space explosion problem* and limits the size of the models that can be analyzed. This section[3] shows how this problem can be approached by exploiting discrete event simulation for model analysis. We present SimQPN - a Java-based simulation tool for QPNs that can be used to analyze QPN models of realistic size and complexity. While doing this, we propose a methodology for simulating QPN models and analyzing the output data from simulation runs. SimQPN can be seen as an implementation of this methodology.

[3] Originally published in Performance Evaluation Journal, Vol. 63, No. 4-5, S. Kounev and A. Buchmann, *SimQPN-a tool and methodology for analyzing queueing Petri net models by means of simulation*, pp. 364-394. Copyright Elsevier (2006).

SimQPN is a discrete-event simulation engine specialized for QPNs. It is extremely lightweight and has been implemented 100% in Java to provide maximum portability and platform-independence. SimQPN simulates QPNs using a sequential algorithm based on the event-scheduling approach for simulation modeling. Being specialized for QPNs, it simulates QPN models directly and has been designed to exploit the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation. Therefore, SimQPN provides much better performance than a general purpose simulator would provide, both in terms of the speed of simulation and the quality of output data provided.

3.1 SimQPN design and architecture

SimQPN has an object-oriented architecture. Every element (for e.g. place, transition or token) of the simulated QPN is internally represented as object. Figure 4 outlines the main simulation routine which drives each simulation run. As already mentioned, SimQPN's internal simulation procedure is based on the event-scheduling approach [Law and Kelton, 2000]. To explain what is understood by event here, we need to look at the way the simulated QPN transitions from one state to another with respect to time. Since only immediate transitions are supported, the only place in the QPN where time is involved is inside the queues of queueing places. Tokens arriving at the queues wait until there is a free server available and are then served. A token's service time distribution determines how long its service continues. After a token has been served it is moved to the depository of the queueing place, which may enable some transitions and trigger their firing. This leads to a change in the marking of the QPN. Once all enabled transitions have fired, the next change of the marking will occur after another service completion at some queue. In this sense, it is the completion of service that initiates each change of the marking. Therefore, we define *event* to be a completion of a token's service at a queue.

SimQPN uses an optimized algorithm for keeping track of the enabling status of transitions. Generally, Petri net simulators need to check for enabled transitions after each change in the marking caused by a transition firing. The exact way they do this, is one of the major factors determining the efficiency of the simulation [Gaeta, 1996]. In [Mortensen, 2001], it is shown how the *locality principle* of colored Petri nets can be exploited to minimize the overhead of checking for enabled transitions. The locality principle states that an occurring transition will only affect the marking on immediate neighbor places, and hence the enabling status of a limited set of neighbor transitions. SimQPN exploits an adaptation of this principle to QPNs, taking into account that tokens deposited into queueing places do not become available for output transitions immediately upon arrival and hence cannot affect the enabling status of the latter. Since checking the enabling status of a transition is a computationally expensive operation, our goal is to make sure that this is done as seldom as possible, i.e., only when there is a real possibility that the status has changed. This translates into the following two cases when the enabling status of a transition needs to be checked:

1. After a change in the token population of an ordinary input place of the transition, as a result of firing of the same or another transition. Three subcases are distinguished:
   (a) Some tokens were added. In this case, it is checked for *newly enabled modes* by considering all modes that are currently marked as disabled and that require tokens of the respective colors added.

(b) Some tokens were removed. In this case, it is checked for *newly disabled modes* by considering all modes that are currently marked as enabled and that require tokens of the respective colors removed.
(c) Some tokens were added and at the same time others were removed. In this case, both of the checks above are performed.

2. After a service completion event at a queueing input place of the transition. The service completion event results in adding a token to the depository of the queueing place. Therefore, in this case, it is only checked for *newly enabled modes* by considering all modes that are currently marked as disabled and that require tokens of the respective color added.

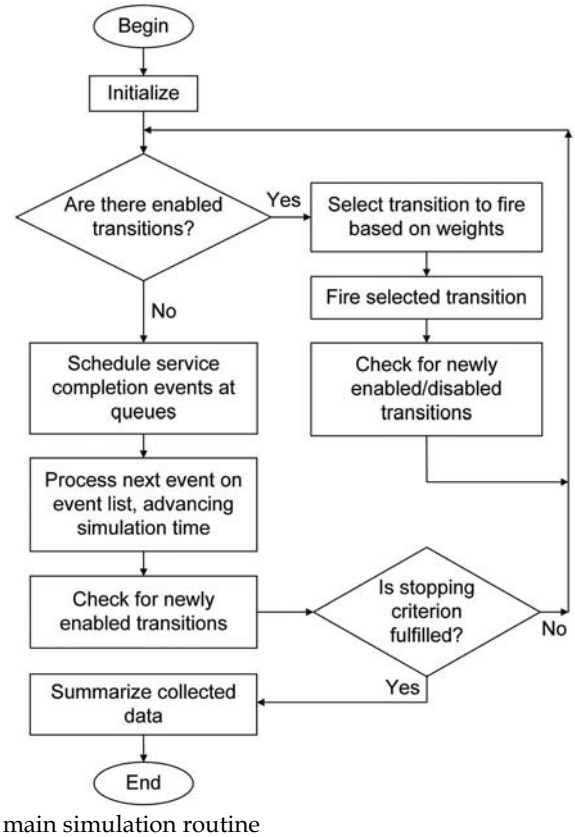


Fig. 4. SimQPN's main simulation routine

SimQPN maintains a global list of currently enabled transitions and for each transition a list of currently enabled modes. The latter are initialized at the beginning of the simulation by checking the enabling status of all transitions. As the simulation progresses, a transition's enabling status is checked only in the above mentioned cases. This reduces CPU costs and speeds up the simulation substantially.

3.2 Output data analysis

SimQPN supports two methods for estimation of the steady state mean residence times of

tokens inside the queues, places and depositories of the QPN. These are the well-known *method of independent replications (IR)* (in its variant referred to as *replication/deletion approach)* and the classical *method of non-overlapping batch means (NOBM).* We refer the reader to [Pawlikowski, 1990; Law and Kelton, 2000; Alexopoulos and Seila, 2001] for an introduction to these methods. Both of them can be used to provide point and interval estimates of the steady state mean token residence time. In cases where one wants to apply a more sophisticated technique for steady state analysis (for example ASAP [Steiger *et al,* 2005]), SimQPN can be configured to output observed token residence times to files (mode 4), which can then be used as input to external analysis tools. Both the replication/deletion approach and the method of non-overlapping batch means have different variants. Below we discuss some details on the way they were implemented in SimQPN.

Replication/Deletion Approach

We briefly discuss the way the replication/ deletion approach is implemented in SimQPN. Suppose that we want to estimate the steady state mean residence time v of tokens of given color at a given place, queue or depository. As discussed in [Alexopoulos and Seila, 2001], in the replication/deletion approach multiple replications of the simulation are made and the average residence times observed are used to derive steady state estimates. Specifically, suppose that n replications of the simulation are made, each of them generating m residence time observations $Y_{i1}, Y_{i2}, \bullet\ \bullet\ \bullet, Y_{im}$. We delete l observations from the beginning of each set to eliminate the initialization bias. The number of observations deleted is determined through the method of Welch [Heidelberger and Welch, 1983]. Let Xi be given by

$$X_i = \frac{\sum_{j=l+1}^{m} Y_{ij}}{m-l} \qquad i = 1, 2, \ldots, n \tag{1}$$

and

$$\overline{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n} \qquad S^2(n) = \frac{\sum_{i=1}^{n} [X_i - \overline{X}(n)]^2}{n-1} \tag{2}$$

Then the X_i's are independent and identically distributed (IID) random variables with $E(X_i) \approx \nu$ and $\overline{X}(n)$ is an approximately unbiased point estimator for v. According to the central limit theorem [Trivedi, 2002], if m is large, the X_i's are going to be approximately normally distributed and therefore the random variable

$$t_n = \frac{[\overline{X}(n) - \nu]}{\sqrt{\frac{S^2(n)}{n}}}$$

will have t distribution with $(n - 1)$ degrees of freedom (df) [Hogg and Craig, 1995] and an approximate $100(1 - \alpha)$ percent confidence interval for v is then given by

$$\overline{X}(n) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{S^2(n)}{n}} \tag{3}$$

where $t_{n-1,1-\alpha/2}$ is the upper $(1 - \alpha/2)$ critical point for the t distribution with $(n - 1)$ df [Pawlikowski, 1990; Trivedi, 2002].

Method of Non-Overlapping Batch Means

Unlike the replication/deletion approach, the method of non-overlapping batch means seeks to obtain independent observations from a single simulation run rather than from

multiple replications. Thus, it has the advantage that it must go through the warm-up period only once and is therefore less sensitive to bias from the initial transient. Suppose that we make a simulation run of length m and then divide the resulting observations $Y_1, Y_2, \bullet\ \bullet\ \bullet, Y_m$ into n batches of length q. Assume that $m = n * q$ and let X_i be the sample (or batch) mean of the q observations in the ith batch, i.e.

$$X_i(q) = \frac{\sum_{j=(i-1)q+1}^{(i-1)q+q} Y_j}{q} \qquad i = 1, 2, \ldots, n \tag{4}$$

The mean v is estimated by $\overline{X}(n) = (\sum_{i=1}^{n} X_i(q))/n$ and it can be shown (see for example [Law and Kelton, 2000]) that an approximate $100(1-\alpha)$ percent confidence interval for v is given by substituting $X_i(q)$ for X_i in Equations (2) and (3) above.

SimQPN offers two different *stopping criteria* for determining how long the simulation should continue. In the first one, the simulation continues until the QPN has been simulated for a user-specified amount of model time *(fixed-sample-size procedure).* In the second one, the length of the simulation is increased sequentially from one checkpoint to the next, until enough data has been collected to provide estimates of residence times with user-specified precision *(sequential procedure).* The precision is defined as an upper bound for the confidence interval half length. It can be specified either as an absolute value (absolute precision) or as a percentage relative to the mean residence time (relative precision). The sequential approach for controlling the length of the simulation is usually regarded as the only efficient way for ensuring representativeness of the samples of collected observations [Law and Kelton, 1982; Heidelberger and Welch, 1983; Pawlikowski *et al,* 1998]. Therefore, hereafter we assume that the sequential procedure is used.

The main problem with the method of non-overlapping batch means is to select the batch size q, such that successive batch means are approximately uncorrelated. Different approaches have been proposed in the literature to address this problem (see for example [Chien, 1994; Alexopoulos & Goldsman, 2004; Pawlikowski, 1990]). In SimQPN, we start with a user-configurable initial batch size (by default 200) and then increase it sequentially until the correlation between successive batch means becomes negligible. Thus, the simulation goes through two stages: the first sequentially testing for an acceptable batch size and the second sequentially testing for adequate precision of the residence time estimates (see Figure 5). The parameters n and p, specifying how often checkpoints are made, can be configured by the user.

We use the *jackknife estimators* [Miller, 1974; Pawlikowski, 1990] of the autocorrelation coefficients to measure the correlation between batch means. A jackknife estimator $\hat{J}(k,q)$ of the autocorrelation coefficient of lag k for the sequence of batch means $X_1(q)$, $X_2(q), \ldots, X_n(q)$ of size q is calculated as follows:

$$\hat{J}(k,q) = 2\hat{r}(k,q) - \frac{\hat{r}'(k,q) + \hat{r}''(k,q)}{2} \tag{5}$$

where $\hat{r}(k,q)$ is the ordinary estimator of the autocorrelation coefficient of lag k, calculated from the formula [Pawlikowski, 1990]:

$$\hat{r}(k,q) = \frac{\frac{1}{n-k}\sum_{i=k+1}^{n}[X_i(q) - \overline{X}(n)][X_{i-k}(q) - \overline{X}(n)]}{\frac{1}{n}\sum_{i=1}^{n}[X_i(q) - \overline{X}(n)]^2} \tag{6}$$

and $\hat{r}'(k,q)$ and $\hat{r}''(k,q)$ are calculated like $\hat{r}(k,q)$, except that $\hat{r}(k,q)$ is the estimator over all n batch means, whereas $\hat{r}'(k,q)$ and $\hat{r}''(k,q)$ are estimators over the first and the second half of the analyzed sequence of n batch means, respectively.

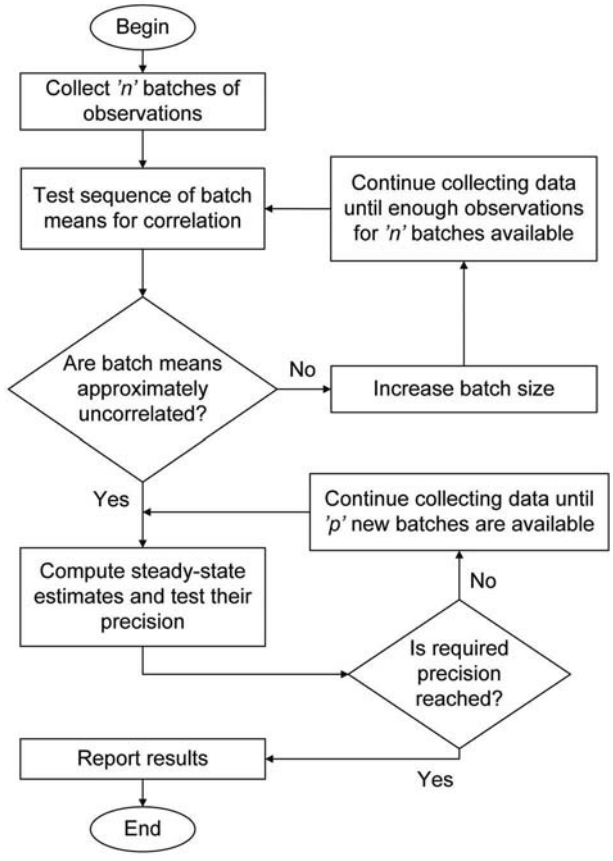


Fig. 5. SimQPN's batch means procedure

We use the algorithm proposed in [Pawlikowski, 1990] to determine when to consider the sequence of batch means for approximately uncorrelated: a given batch size is accepted to yield approximately uncorrelated batch means if all autocorrelation coefficients of lag k ($k = 1, 2, \ldots, L$; where $L = 0.1 * n$) are statistically negligible at a given significance level β_k, $0 < \beta_k < 1$. To get an acceptable overall significance level β we assume that

$$\beta < \sum_{k=1}^{L} \beta_k \tag{7}$$

As recommended in [Pawlikowski, 1990], in order to get reasonable estimators of the autocorrelation coefficients, we apply the above batch means correlation test only after at least 100 batch means have been recorded (i.e., n >= 100). In fact, by default n is set to 200 in SimQPN. Also to ensure approximate normality of the batch means, the initial batch

size (i.e., the minimal batch size) is configured to 200.

SimQPN Validation

We have validated the algorithms implemented in SimQPN by subjecting them to a rigorous experimental analysis and evaluating the quality of point and interval estimates [Kounev and Buchmann, 2006]. In particular, the variability of point estimates provided by SimQPN and the coverage of confidence intervals reported were quantified. A number of different models of realistic size and complexity were considered. Our analysis showed that data reported by SimQPN is very accurate and stable. Even for residence time, the metric with highest variation, the standard deviation of point estimates did not exceed 2.5% of the mean value. In all cases, the estimated coverage of confidence intervals was less than 2% below the nominal value (higher than 88% for 90% confidence intervals and higher than 93% for 95% confidence intervals).

4. Performance modeling and analysis of distributed systems

Queueing Petri nets are a powerful formalism that can be exploited for modeling distributed systems and analyzing their performance and scalability. However, building models that accurately capture the different aspects of system behavior is a very challenging task when applied to realistic systems. In this section[4], we present a case study in which QPNs are used to model a real-life system and analyze its performance and scalability. In parallel to this, we present a practical performance modeling methodology for distributed systems which helps to construct models that accurately reflect the performance and scalability characteristics of the latter. Our methodology builds on the methodologies proposed by Menascé, Almeida & Dowdy in [Menascé *et al,* 1994; 1999; Menascé & Almeida, 1998; 2000; Menascé *et al,* 2004], however, a major difference is that our methodology is based on QPN models as opposed to conventional queueing network models and it is specialized for distributed component-based systems. The system studied is a deployment of the industry-standard SPECjAppServer2004 benchmark. A detailed model of the system and its workload is built in a step-by-step fashion. The model is validated and used to predict the system performance for several deployment configurations and workload scenarios of interest. In each case, the model is analyzed by means of simulation using SimQPN. In order to validate the approach, the model predictions are compared against measurements on the real system. In addition to CPU and I/O contention, it is demonstrated how some more complex aspects of system behavior, such as thread contention and asynchronous processing, can be modeled.

4.1 The SPECjAppServer2004 benchmark

SPECjAppServer2004 is a new industry-standard benchmark for measuring the performance and scalability of J2EE hardware and software platforms. It implements a representative workload that exercises all major services of the J2EE platform in a

[4] Portions reprinted, with permission, from IEEE Transactions on Software Engineering, Vol. 32, No. *7, Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets,* pp. 486-502. (c) [2006] IEEE.

complete *end-to-end* application scenario. The SPECjAppServer2004 workload has been specifically modeled after an automobile manufacturer whose main customers are automobile dealers [SPEC, 2004]. Dealers use a Web-based user interface to browse an automobile catalogue, purchase automobiles, sell automobiles and track their inventory. As depicted in Figure 6, SPECjAppServer2004's business model comprises five domains: customer domain dealing with customer orders and interactions, dealer domain offering Web-based interface to the services in the customer domain, manufacturing domain performing "just in time" manufacturing operations, supplier domain handling interactions with external suppliers, and corporate domain managing all dealer, supplier and automobile information.

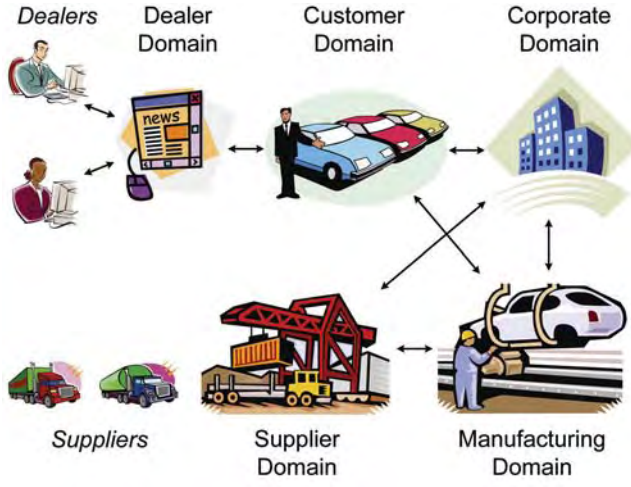


Fig. 6. SPECjAppServer2004 business model.

The customer domain hosts an *order entry application* that provides some typical online ordering functionality. Orders for more than 100 automobiles are called *large orders.* The dealer domain hosts a Web application (called *dealer application)* that provides a Web-based interface to the services in the customer domain. The manufacturing domain hosts a *manufacturing application* that models the activity of production lines in an automobile manufacturing plant. There are two types of production lines, planned lines and large order lines. Planned lines run on schedule and produce a predefined number of automobiles. Large order lines run only when a large order is received in the customer domain. The unit of work in the manufacturing domain is a *work order.* Each work order moves along three virtual stations, which represent distinct operations in the manufacturing flow. In order to simulate activity at the stations, the manufacturing application waits for a designated time (333 ms) at each station. Once the work order is complete, it is marked as completed and inventory is updated. When the inventory of parts gets depleted, suppliers need to be located and purchase orders need to be sent out. This is done by contacting the supplier domain, responsible for interactions with external suppliers.

4.2 Motivation

Consider an automobile manufacturing company that wants to use e-business technology to support its order-inventory, supply-chain and manufacturing operations. The company has decided to employ the J2EE platform and is in the process of developing a J2EE application. Let us assume that the first prototype of this application is SPECjAppServer2004 and that the company is testing the application in the deployment environment depicted in Figure 7. This environment uses a cluster of WebLogic servers (WLS) as a J2EE container and an Oracle database server (DBS) for persistence. We assume that all servers in the WebLogic cluster are identical and that initially only two servers are available. The company is now about to conduct a performance modeling study of their system in order to evaluate its performance and scalability. In the following, we present a practical performance modeling methodology in a step-by-step fashion showing how each step is applied to the considered scenario.

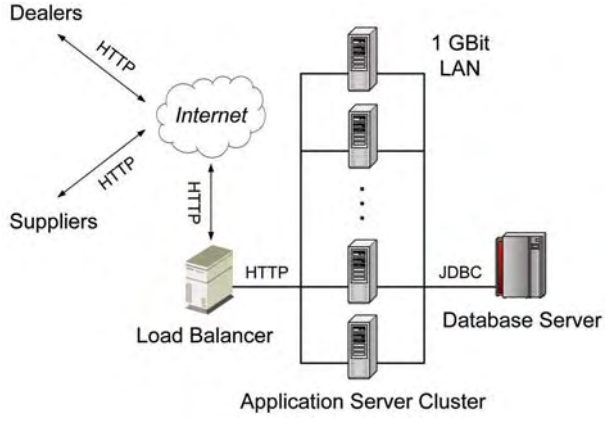


Fig. 7. Deployment environment.

4.3 Step 1: Establish performance modeling objectives

Let us assume that under peak conditions, 152 concurrent dealer clients (100 Browse, 26 Purchase and 26 Manage) are expected and the number of planned production lines could increase up to 100. Moreover, the workload is forecast to grow by 300% over the next 5 years. The average *dealer think time* is 5 seconds, i.e., the time a dealer "thinks" after receiving a response from the system before sending a new request. On average 10 percent of all orders placed are assumed to be large orders. The average delay after completing a work order at a planned production line before starting a new one is 10 seconds. Note that all of these numbers were chosen arbitrarily in order to make our motivating scenario more specific. Based on these assumptions, the following concrete goals are established:

- Predict the performance of the system under peak operating conditions with 6 WebLogic servers. What would be the average throughput and response time of dealer transactions and work orders? What would be the CPU utilization of the servers?
- Determine if 6 WebLogic servers would be enough to ensure that the average response times of business transactions do not exceed half a second. Predict how much system performance would improve if the load balancer is upgraded with

a slightly faster CPU.

- Study the scalability of the system as the workload increases and additional WebLogic servers are added. Determine which servers would be most utilized under heavy load and investigate if they are potential bottlenecks.

4.4 Step 2: Characterize the system in its current state

As shown in Figure 7, the system we are considering has a two-tier hardware architecture consisting of an application server tier and a database server tier. Incoming requests are evenly distributed across the nodes in the application server cluster. For HTTP requests, this is achieved using a software load balancer running on a dedicated machine. For RMI requests, this is done transparently by the EJB client stubs. Table 1 describes the system components in terms of the hardware and software platforms used. This information is enough for the purposes of our study.

Component	Description
Load Balancer	Commercial HTTP Load Balancer 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
App. Server Cluster Nodes	WebLogic 8.1 Server 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
Database Server	Oracle 9i Server 2 x AMD Athlon MP2000+ CPU 2 GB RAM, SuSE Linux 8
Local Area Network	1 GBit Switched Ethernet

Table 1. System component details

4.5 Step 3: Characterize the workload

Identify the Basic Components of the Workload

As discussed in Section 4.1, the SPECjAppServer2004 benchmark application is made up of three major subapplications - the dealer application, the order entry application and the manufacturing application. The dealer and order entry applications process business transactions of three types - Browse, Purchase and Manage. Hereafter, the latter are referred to as *dealer transactions.* The manufacturing application, on the other hand, is running production lines which process work orders. Thus, the SPECjAppServer2004 workload is composed of two basic components: dealer transactions and work orders.

Partition Basic Components into Workload Classes

There are three types of dealer transactions and since we are interested in their individual behavior we model them using separate workload classes. Work orders, on the other hand, can be divided into two types based on whether they are processed on a planned or large order line. Planned lines run on schedule and complete a predefined number of work orders per unit of time. In contrast, large order lines run only when a large order arrives in the customer domain. Each large order generates a separate work order processed *asynchronously* on a dedicated large order line. Thus, work orders originating from large orders are different from ordinary work orders in terms of the way their processing is initiated and in terms of their resource usage. To distinguish between the two types of work orders, they are modeled using two separate workload classes:

WorkOrder (for ordinary work orders) and *LargeOrder* (for work orders generated by large orders). Altogether, we end up with five workload classes: Browse, Purchase, Manage, WorkOrder and LargeOrder.

Identify the System Components and Resources Used by Each Workload Class

The following hardware resources are used by dealer transactions: CPU of the load balancer machine (LB-C), CPU of an application server in the cluster (AS-C), CPUs of the database server (DB-C), disk drive of the database server (DB-D), Local Area Network (LAN). WorkOrders and LargeOrders use the same resources with exception of the first one, since their processing is driven through direct RMI calls to the EJBs in the WebLogic cluster, bypassing the HTTP load balancer. As far as software resources are concerned, all workload classes use the WebLogic servers and the Oracle DBMS. Dealer transactions additionally use the HTTP load balancer, which is running on a dedicated machine.

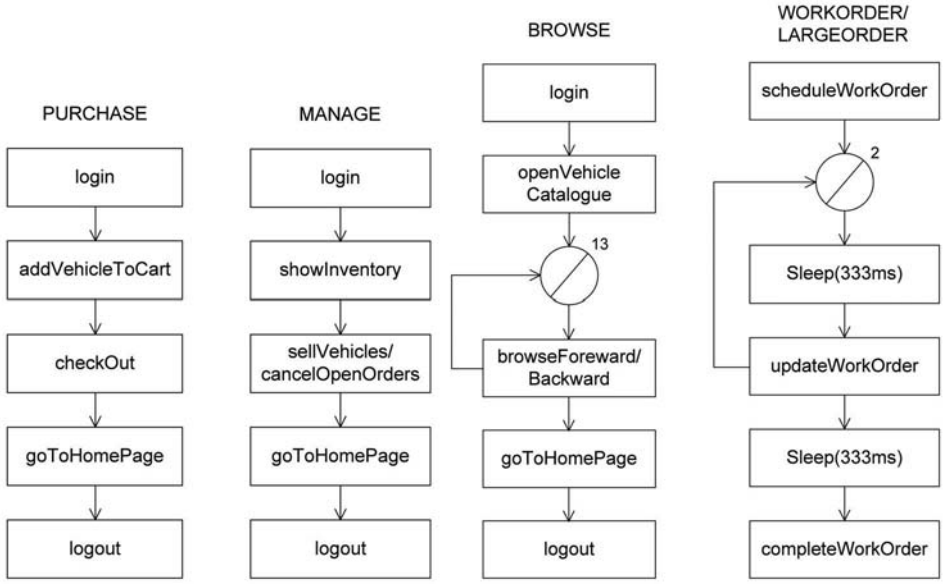


Fig. 8. Execution graphs for Purchase, Manage, Browse, WorkOrder and LargeOrder.

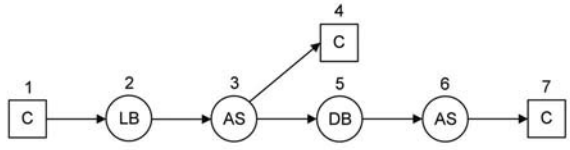
Describe the Inter-Component Interactions and Processing Steps for Each Workload Class

All of the five workload classes identified represent composite transactions. Figure 8 uses execution graphs to illustrate the subtransactions (processing steps) of transactions from the different workload classes. For every subtransaction (represented as a rectangle) multiple system components are involved and they interact to perform the respective operation. The inter-component interactions and flow of control during the processing of subtransactions are depicted in Figure 9 by means of client/server interaction diagrams. Directed arcs show the flow of control from one node to the next during execution. Depending on the path followed, different execution scenarios are possible. For example, for dealer subtransactions two scenarios are possible depending on whether the database needs to be accessed or not. Dealer subtransactions that do not access the database (e.g., goToHomePage) follow the path 1→2→3→4, whereas dealer subtransactions that access

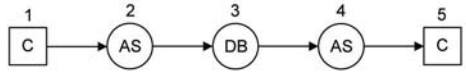
the database (e.g., showlnven-tory) follow the path 1→2→3→5→6→7. Since most dealer subtransactions do access the database, for simplicity, it is assumed that all of them follow the second path.

Characterize Workload Classes in Terms of Their Service Demands and Workload Intensity

Since the system is available for testing, the service demands can be determined by injecting load into the system and taking measurements. Note that it is enough to have a single WebLogic server available in order to do this, i.e., it is not required to have a realistic production like testing environment. For each of the five workload classes a separate experiment was conducted injecting transactions from the respective class and measuring the utilization of the various system resources. CPU utilization was measured using the vmstat utility on Linux. The disk utilization of the database server was measured with the help of the Oracle 9i Intelligent Agent, which proved to have negligible overhead. Service demands were derived using the Service Demand Law [Menasce and Almeida, 1998]. Table 2 reports the service demand parameters for the five workload classes. It was decided to ignore the network, since all communications were taking place over 1 GBit LAN and communication times were negligible.



(A). Subtransactions of Browse, Purchase and Manage



(B). Subtransactions of WorkOrder and LargeOrder

Fig. 9. Client/server interaction diagrams for Subtransactions.

Workload Class	LB-C	AS-C	DB-C	DB-D
Browse	42.72ms	130ms	14ms	5ms
Purchase	9.98ms	55ms	16ms	8ms
Manage	9.93ms	59ms	19ms	7ms
WorkOrder	-	34ms	24ms	2ms
LargeOrder	-	92ms	34ms	2ms

Table 2. Workload service demand parameters

In order to keep the workload model simple, it is assumed that the total service demand of a transaction at a given system resource is spread evenly over its subtransactions. Thus, the service demand of a subtransaction can be estimated by dividing the measured total service demand of the transaction by the number of subtransactions it has. It is also assumed that all service demands are exponentially distributed. Whether these simplifications are acceptable will become clear later when the model is validated. In case the estimation proves to be too inaccurate, one might have to come back and refine the

workload model by measuring the service demands of subtransactions individually.
Now that the service demands of workload classes have been quantified, the workload intensity must be specified. For each workload class, the number of transactions that contend for system resources must be indicated. The way workload intensity is specified is dictated by the modeling objectives. In our case, workload intensity was defined in terms of the following parameters (see Section 4.3):

- Number of concurrent dealer clients of each type and the average dealer think time.
- Number of planned production lines and the average time they wait after processing a WorkOrder before starting a new one *(manufacturing think time* or *mfg think time).*

The concerete values of the above parameters under peak operating conditions were given in Section 4.3. The workload, however, had been forecast to grow by 300% and another goal of the study was to investigate the scalability of the system as the load increases. Therefore, scenarios with up to 3 times higher workload intensity need to be considered as well.

4.6 Step 4: Develop a performance model

A QPN model of the system under study is now built and then customized to the concrete configurations of interest. We start by discussing the way basic components of the workload are modeled. During workload characterization, five workload classes were identified. All of them represent composite transactions and are modeled using the following token types (colors): 'B' for Browse, 'P' for Purchase, 'M' for Manage, 'W' for WorkOrder and 'L' for Large-Order. The subtransactions of transactions from the different workload classes were shown in Figure 8. In order to make the performance model more compact, it is assumed that each server used during processing of a subtransaction is visited only once and that the subtransaction receives all of its service demands at the server's resources during that single visit. This simplification is typical for queueing models and has been widely employed. While characterizing the workload service demands in Section 4.5, we additionally assumed that the total service demand of a transaction at a given system resource is spread evenly over its subtransactions. This allows us to consider the subtransactions of a given workload class as equivalent in terms of processing behavior and resource consumption. Thus, we can model subtransactions using a single token type (color) per workload class as follows: 'b' for Browse, 'p' for Purchase, 'm' for Manage, 'w' for WorkOrder and 'l' for LargeOrder. For the sake of compactness, the following additional notation will be used:
Symbol 'D' will denote a 'B', 'P' or 'M' token, i.e., token representing a dealer transaction.
Symbol 'd' will denote a 'b', 'p' or 'm' token, i.e., token representing a dealer subtransaction.
Symbol 'o' will denote a 'b', 'p', 'm', 'w' or 'l' token, i.e., token representing a subtransaction of arbitrary type, hereafter called *subtransaction token.*
To further simplify the model, we assume that LargeOrder transactions are executed with a single subtransaction, i.e., their four subtransactions are bundled into a single subtransaction. The effect of this simplification on the overall system behavior is negligible, because large orders constitute only 10 percent of all orders placed, i.e., relatively small portion of the system workload. Mapping the system components, resources and inter-component interactions to QPN models constructs, we arrive at the model depicted in Figure 10. We use the notation "$A\{x\} \rightarrow B\{y\}$" to denote a firing mode

in which an 'x' token is removed from place A and a 'y' token is deposited in place B. Similarly, "$A\{x\} \rightarrow \{\}$" means that an 'x' token is removed from place A and destroyed without depositing tokens anywhere. Table 3 provides some details on the places used in the model.

All token service times at the queues of the model are assumed to be exponentially distributed. We now examine in detail the life-cycle of tokens in the QPN model. As already discussed, upper-case tokens represent transactions, whereas lower-case tokens represent subtransactions. In the initial marking, tokens exist only in the depositories of places C_1 and C_2 The initial number of 'D' tokens ('B', 'P' or 'M') in the depository of the former determines the number of concurrent dealer clients, whereas the initial number of 'W' tokens in the depository of the latter determines the number of planned production lines running in the manufacturing domain.

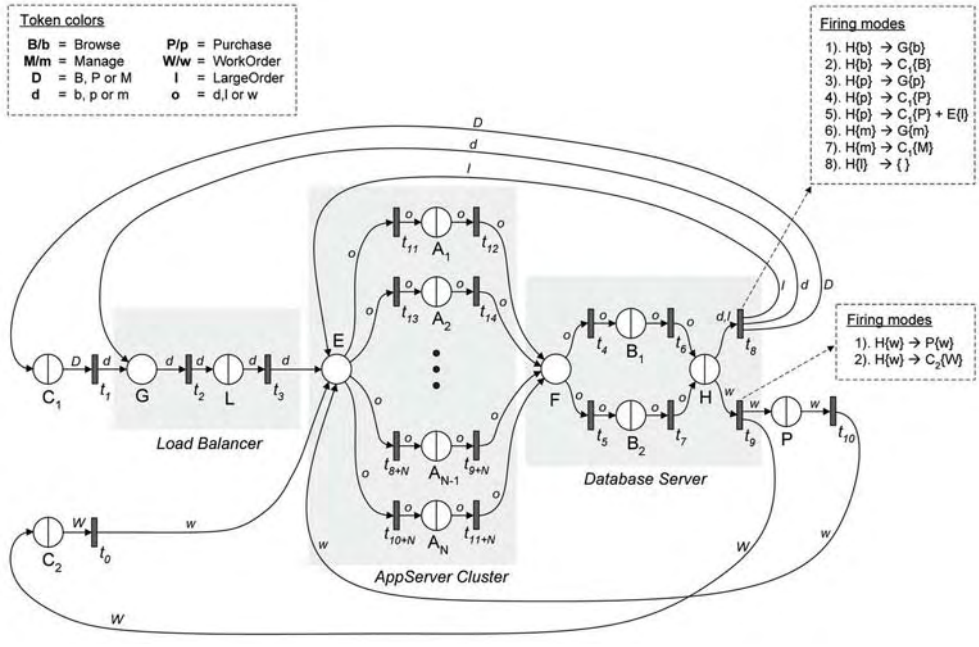


Fig. 10. QPN model of the system.

When a dealer client starts a dealer transaction, transition t_1 is fired destroying a 'D' token from the depository of place C_1 and creating a 'd' token in place *G*, which corresponds to starting the first subtransaction. The flow of control during processing of subtransactions in the system is modeled by moving their respective subtransaction tokens across the different places of the QPN. Starting at place *G*, a dealer subtransaction token ('d') is first sent to place *L* where it receives service at the CPU of the load balancer. After that it is moved to place *E* and from there it is routed to one of the *N* application server CPUs represented by places A_1 to A_N Transitions $t_{11}, t_{13}, \ldots, t_{10+N}$ have equal firing probabilities (weights), so that subtransactions are probabilistically load-balanced across the N application servers. This approximates the round-robin mechanism used by the load-balancer to distribute incoming requests among the servers. Having completed its service

at the application server CPU, the dealer subtransaction token is moved to place *F* from where it is sent to one of the two database server CPUs with equal probability (transitions t4 and t_5 have equal firing weights). After completing its service at the CPU, the dealer subtransaction token is moved to place *H* where it receives service from the database disk subsystem. Once this is completed, the dealer subtransaction token is destroyed by transition t_8 and there are two possible scenarios:

1. A new 'd' token is created in place *G*, which starts the next dealer subtransaction.
2. If there are no more subtransactions to be executed, the 'D' token removed from place C_1 in the beginning of the transaction is returned. If the completed transaction is of type Purchase and it has generated a large order, additionally a token 'l' is created in place *E*.

Note that, since LargeOrder transactions are assumed to be executed with a single subtransaction, to simplify the model, we create the subtransaction token ('l') directly instead of first creating a transaction token ('L'). So, in practice, 'L' tokens are not used explicitly in the model. After a 'D' token of a completed transaction returns back to place C1, it spends some time at the IS queue of the latter. This corresponds to the dealer think time. Once the dealer think time has elapsed, the 'D' token is moved to the depository and the next transaction is started.

Place	Tokens	Queue Type	Description
C_1	{B,P,M}	$G/M/\infty/IS$	Queueing place used to model concurrent dealer clients conducting dealer transactions. The time tokens spend here corresponds to the dealer think time.
C_2	{W}	$G/M/\infty/IS$	Queueing place used to model planned production lines driving work order processing. The time tokens spend here corresponds to the mfg think time.
G	{b,p,m}	*na*	Ordinary place where dealer subtransaction tokens are created when new subtransactions are started.
L	{b,p,m}	$G/M/1/PS$	Queueing place used to model the CPU of the load balancer machine.
E	{b,p,m,l,w}	*na*	Ordinary place where subtransaction tokens arrive before they are distributed over the application server nodes.
A_i	{b,p,m,l,w}	$G/M/1/PS$	Queueing places used to model the CPUs of the N application server nodes.
F	{b,p,m,l,w}	*na*	Ordinary place where subtransaction tokens arrive when visiting the database server. From here tokens are evenly distributed over the two database server CPUs.
B_j	{b,p,m,l,w}	$G/M/1/PS$	Queueing places used to model the two CPUs of the database server.
H	{b,p,m,l,w}	$G/M/1/FCFS$	Queueing place used to model the disk subsystem (made up of a single 100 GB disk drive) of the database server.
P	{w}	$G/M/\infty/IS$	Queueing place used to model the virtual production line stations that work orders move along during their processing. The time tokens spend here corresponds to the average delay at a production line station (i.e., 333 ms) emulated by the manufacturing application during work order processing.

Table 3. Places used in the QPN model

When a WorkOrder transaction is started on a planned line in the manufacturing domain, transition t_o is fired destroying a 'W' token from the depository of place C_2 and creating a 'w' token in place E, which corresponds to starting the first subtransaction. Since WorkOrder subtransaction requests are load-balanced transparently (by the EJB client stubs) without using a load balancer, the WorkOrder subtransaction token ('w') is routed directly to the application server CPUs - places A_1 to A_N. It then moves along the places representing the application server and database server resources exactly in the same way as dealer subtransaction tokens. After it completes its service at place H, the following two scenarios are possible:

1. The 'w' token is sent to place P whose IS queue delays it for 333 ms, corresponding to the delay at a virtual production line station. After that the token is destroyed by transition t_{10} and a new 'w' token is created in place E, representing the next WorkOrder subtransaction.
2. If there are no more subtransactions to be executed, the 'w' token is destroyed by transition t_9 and the 'W' token removed from place C_2 in the beginning of the transaction is returned.

After a 'W' token of a completed transaction returns back to place C_2, it spends some time at the IS queue of the latter. This corresponds to the time waited after completing a work order at a production line before starting the next one. Once this time has elapsed, the 'W' token is moved to the depository and the next transaction is started.

All transitions of the model are immediate and their firing modes, except for transitions t_0, t_1, t_8 and t_9, are defined in such a way that whenever they fire they simply move a token from their input place to their output place. Transitions t_0 and t_1 have similar behavior except that when they remove an upper case token from their input place they deposit the respective lower case token into the output place. We assign the same firing weight (more specifically 1) to all modes of these transitions, so that they have the same probability of being fired when multiple of them are enabled at the same time. The definition of the firing modes of transitions t_8 and t_9 is a little more complicated. The firing modes are described in Tables 4 and 5, respectively. The assignment of weights to the modes of these transitions is critical to achieving the desired behavior of transactions in the model. Weights must be assigned in such a way that transactions are terminated only after all of their subtransactions have been completed. We will now explain how this is done, starting with transition t_9 since this is the simpler case. According to Section 4.5 (Figure 8), WorkOrder transactions are comprised of four subtransactions. This means that, for every WorkOrder transaction, four subtransactions have to be executed before the transaction is completed. To model this behavior, the firing weights (probabilities) of modes 1 and 2 are set to 3/4 and 1/4, respectively. Thus, out of every four times a 'w' token arrives in place H and enables transition t_9, on average the latter will be fired three times in mode 1 and one time in mode 2, completing a WorkOrder transaction. Even though the order of these firings is not guaranteed, the resulting model closely approximates the real system in terms of resource consumption and queueing behavior.

Transition t_8, on the other hand, has eight firing modes as shown in Table 4. According to Section 4.5 (Figure 8), Browse transactions have 17 subtransactions, whereas Purchase and Manage have only 5. This means that, for every Browse transaction, 17 subtransactions have to be executed before the transaction is completed, i.e., out of every 17 times a 'b' token arrives in place H and enables transition $t8$, the latter has to be fired 16 times in

mode 1 and one time in mode 2 completing a Browse transaction. Out of every 5 times a 'p' token arrives in place *H* and enables transition *t8*, the latter has to be fired 4 times in mode 3 and one time in mode 4 or mode 5, depending on whether a large order has been generated. On average 10% of all completed Purchase transactions generate large orders. Modeling these conditions probabilistically leads to a system of simultaneous equations that the firing weights (probabilities) of transition *t8* need to fulfil. One possible solution is the following: $w(1) = 16$, $w(2) = 1$, $w(3) = 13.6$, $w(4) = 3.06$, $w(5) = 0.34$, $w(6) = 13.6$, $w(7) = 3.4$, $w(8) = 17$.

Mode	Action	Case Modeled
1	$H\{b\} \rightarrow G\{b\}$	Browse subtransaction has been completed. Parent transaction is not finished yet.
2	$H\{b\} \rightarrow C_1\{B\}$	Browse subtransaction has been completed. Parent transaction is finished.
3	$H\{p\} \rightarrow G\{p\}$	Purchase subtransaction has been completed. Parent transaction is not finished yet.
4	$H\{p\} \rightarrow C_1\{P\}$	Purchase subtransaction has been completed. Parent transaction is finished.
5	$H\{p\} \rightarrow C_1\{P\} + E\{l\}$	Same as (4), but assuming that completed transaction has generated a large order.
6	$H\{m\} \rightarrow G\{m\}$	Manage subtransaction has been completed. Parent transaction is not finished yet.
7	$H\{m\} \rightarrow C_1\{M\}$	Manage subtransaction has been completed. Parent transaction is finished.
8	$H\{l\} \rightarrow \{\}$	LargeOrder transaction has been completed. Its token is simply destroyed.

Table 4. Firing modes of transition t_8

Mode	Action	Case Modeled
1	$H\{w\} \rightarrow P\{w\}$	WorkOrder subtransaction has been completed. Parent transaction is not finished yet.
2	$H\{w\} \rightarrow C_2\{W\}$	WorkOrder subtransaction has been completed. Parent transaction is finished.

Table 5. Firing modes of transition t_9

The workload intensity and service demand parameters from Section 4.5 are used to provide values for the service times of tokens at the various queues of the model. A separate set of parameter values is specified for each workload scenario considered. The service times of subtransactions at the queues of the model are estimated by dividing the total service demands of the respective transactions by the number of subtransactions they have.

4.7 Step 5: Validate, refine and/or calibrate the model

The model developed in the previous sections was validated by comparing its predictions against measurements on the real system. Two application server nodes were available for the validation experiments. The model predictions were verified for a number of different scenarios under different transaction mixes and workload intensities. The model was analyzed by means of simulation using SimQPN. The method of non-overlapping batch means was used for steady state analysis. Both the variation of point estimates from

multiple runs of the simulation and the variation of measured performance metrics from multiple tests were negligible. For all metrics, the standard deviation of estimates was less than 2% of the respective mean value. The metrics considered were transaction throughput (X_i), transaction response time (R_i) and server utilization (U_{LB} for the load balancer, U_{AS} for the application servers and U_{DB} for the database server). The maximum modeling error for throughput was 9.3%, for utilization 9.1% and for response time 12.9%. Varying the transaction mix and workload intensity led to predictions of similar accuracy. However, even though the model was deemed valid at this point of the study, as we will see later, the model can lose its validity when it is modified in order to reflect changes in the system.

4.8 Step 6: Use model to predict system performance

In Section 4.3 some concrete goals were set for the performance study. The system model is now used to predict the performance of the system for the deployment configurations and workload scenarios of interest. In order to validate our approach, for each scenario considered, we will compare the model predictions against measurements on the real system. Note that this validation is not part of the methodology itself and normally it does not have to be done. Indeed, if we would have to validate the model results for every scenario considered, there would be no point in using the model in the first place. The reason we validate the model results here is to demonstrate the effectiveness of our modeling approach and showcase the predictive power of the QPN models it is based on.

As in the validation experiments, for all scenarios considered in this section, the model is analyzed by means of simulation using SimQPN and the method of non-overlapping batch means is used for steady state analysis. Both the variation of point estimates from multiple runs of the simulation and the variation of measured performance metrics from multiple tests are negligible. For all metrics, the standard deviation of estimates is less than 2% of the respective mean value. Table 7 shows the model predictions for two scenarios under peak conditions with 6 application server nodes. The first one uses the original load balancer, while the second one uses an upgraded load balancer with a faster CPU. The faster CPU results in lower service demands as shown in Table 6. With the original load balancer, six application server nodes turned out to be insufficient to guarantee average response times of business transactions below half a second. However, with the upgraded load balancer this was achieved. In the rest of the scenarios considered, the upgraded load balancer will be used.

Load Balancer	Browse	Purchase	Manage
Original	42.72ms	9.98ms	9.93ms
Upgraded	32.25ms	8.87ms	8.56ms

Table 6. Load balancer service demands

We now consider the behavior of the system as the workload intensity increases beyond peak conditions and further application server nodes are added. Table 8 shows the model predictions for two scenarios with an increased number of concurrent Browse clients, i.e., 150 in the first one and 200 in the second one. In both scenarios the number of application server nodes is 8. As evident from the results, the load balancer is completely saturated when increasing the workload intensity and it becomes a bottleneck limiting the overall system performance. Therefore, adding further application server nodes would not bring

any benefit, unless the load balancer is replaced with a faster one.

	Original Load Balancer			Upgraded Load Balancer		
METRIC	Model (99% c.i.)	Msrd.	Error	Model (99% c.i.)	Msrd.	Error
X_B	17.879 (+/- 0.805)	17.742	+0.8%	18.444 (+/- 0.646)	18.347	+0.5%
X_P	5.005 (+/- 0.225)	4.913	+1.9%	4.995 (+/- 0.175)	5.072	-1.5%
X_M	5.006 (+/- 0.227)	4.995	-0.2%	5.035 (+/- 0.176)	5.032	+0.1%
X_W	9.034 (+/- 0.407)	8.880	+1.7%	9.002 (+/- 0.315)	8.850	+1.7%
X_L	0.519 (+/- 0.023)	0.490	+5.9%	0.505 (+/- 0.018)	0.515	-1.9%
R_B	568ms (+/- 14.1ms)	534ms	+6.4%	418ms (+/- 6.9ms)	440ms	-5.0%
R_P	213ms (+/- 6.2ms)	198ms	+7.6%	182ms (+/- 4.6ms)	165ms	+10.3%
R_M	227ms (+/- 6.5ms)	214ms	+6.1%	196ms (+/- 4.6ms)	187ms	+4.8%
R_W	1112ms (+/- 2.3ms)	1135ms	-2.0%	1115ms (+/- 2.4ms)	1123ms	-0.7%
U_{LB}	86.7% (+/- 0.4)	88.0%	-1.5%	68.6% (+/- 0.3)	70.0%	-2.0%
U_{AS}	54.2% (+/- 0.2)	53.8%	+0.7%	55.7% (+/- 0.2)	55.3%	+0.7%
U_{DB}	33.0% (+/- 0.2)	34.5%	-4.3%	33.6% (+/- 0.2)	35.0%	-4.0%

Table 7. Analysis results for scenarios under peak conditions with 6 app. server nodes

	Heavy Load Scenario 1			Heavy Load Scenario 2		
METRIC	Model (99% c.i.)	Msrd.	Error	Model (99% c.i.)	Msrd.	Error
X_B	26.419 (+/- 1.189)	25.905	+2.0%	28.414 (+/- 0.994)	26.987	+5.3%
X_P	4.936 (+/- 0.222)	4.817	+2.5%	4.606 (+/- 0.161)	4.333	+6.3%
X_M	4.935 (+/- 0.220)	4.825	+2.3%	4.610 (+/- 0.162)	4.528	+1.8%
X_W	8.989 (+/- 0.405)	8.820	+1.9%	8.963 (+/- 0.314)	8.970	+0.1%
X_L	0.508 (+/- 0.023)	0.488	+4.1%	0.455 (+/- 0.016)	0.417	+9.1%
R_B	655ms (+/- 14.5ms)	714ms	-8.3%	2043ms (+/- 31.5ms)	2288ms	-10.7%
R_P	250ms (+/- 7.6ms)	257ms	-2.7%	637ms (+/- 14.3ms)	802ms	-20.6%
R_M	259ms (+/- 7.9ms)	276ms	-6.2%	633ms (+/- 14.2ms)	745ms	-15.0%
R_W	1116ms (+/- 2.1ms)	1128ms	-1.1%	1123ms (+/- 2.3ms)	1132ms	-0.8%
U_{LB}	93.8% (+/- 0.2)	95.0%	-1.3%	99.9% (+/- 0.1)	100.0%	0.0%
U_{AS}	54.2% (+/- 0.4)	54.1%	+0.2%	57.3% (+/- 0.3)	55.7%	+2.9%
U_{DB}	38.8% (+/- 0.2)	42.0%	-7.6%	39.6% (+/- 0.2)	42.0%	-5.7%

Table 8. Analysis results for scenarios under heavy load with 8 app. server nodes

4.9 Modeling thread contention

Since the load balancer is the bottleneck resource, it is interesting to investigate its behavior a little further. Until now it was assumed that when a request arrives at the load balancer, there is always a free thread which can start processing it immediately. However, if one keeps increasing the workload intensity, the number of concurrent requests at the load balancer will eventually exceed the number of available threads. The latter would lead to thread contention, resulting in additional delays at the load balancer, not captured by our system model. This is a typical example how a valid model may lose its validity as the workload evolves. We will now show how the model can be refined to capture the thread contention at the load balancer.

Extending the System Model

In Figure 11, an extended version of our system model is shown, which includes an ordinary place *T* representing the load balancer thread pool. Before a dealer request is scheduled for processing at the load balancer CPU, a token 't' representing a thread is

allocated from the thread pool. After the dealer request has been served at the load balancer CPU, the token is returned back to the pool. Thus, if an arriving request finds no available thread, it will have to wait in place *G* until a thread is released. The initial population of place *T* determines the number of threads in the thread pool. At first sight, this appears to be the right approach to model the thread contention at the load balancer. However, an attempt to validate the extended model reveals a significant discrepancy between the model predictions and measurements on the real system. In particular, it stands out that predicted response times are much lower than measured response times for dealer transactions with low workload intensities. A closer investigation shows that the problem is in the way dealer subtransaction tokens arriving in place *G* are scheduled for processing at the load balancer CPU. Dealer sub-transaction tokens become available for firing of transition t2 immediately upon their arrival at place *G*. Thus, whenever arriving tokens are blocked in place *G* their order of arrival is lost. After a thread is released, transition t2 fires in one of its enabled modes with equal probability. Therefore, the order in which waiting subtransaction tokens are scheduled for processing does not match the order of their arrival at place *G*. This obviously does not reflect the way the real system works and renders the model unrepresentative.

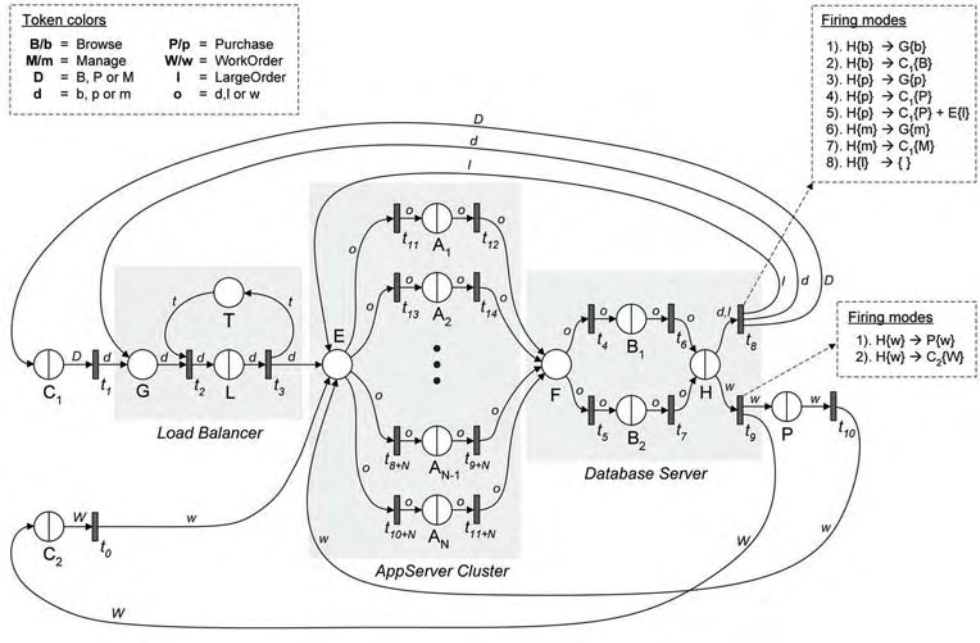


Fig. 11. Extended QPN model of the system (capturing thread contention at the load balancer).

Introducing QPN Departure Disciplines

The above situation describes a common drawback of Petri net models, i.e., tokens inside ordinary places are not distinguished in terms of their order of arrival. One approach to address the problem would be to replace the ordinary place *G* with an immediate

queueing place containing a FCFS queue. However, simply using a FCFS queue would not resolve the problem since arriving tokens would be served immediately and moved to the depository where their order of arrival will still be lost. To address this, we could exploit the generalized queue definition in [Bause, 1993] to define the scheduling strategy of place *G's* queue in such a way that tokens are served immediately according to FCFS, but only if the depository is empty. If there is a token in the depository, all tokens are blocked in their current position until the depository becomes free. Even though this would theoretically address the issue with the token order, it would create another problem. The available tools and techniques for QPN analysis, including SimQPN, do not support queues with scheduling strategy dependent on the state of the depository. Indeed, the generalized queue definition given in [Bause, 1993], while theoretically powerful, is impractical to implement, so in practice it is rarely used and queues in QPNs are usually treated as conventional queues from queueing network theory. The way we address the problem is by introducing *departure disciplines,* which are a simple yet powerful feature we have added to SimQPN. The departure discipline of an ordinary place or depository determines the order in which arriving tokens become available for output transitions. We define two departure disciplines, Normal (used by default) and First-In-First-Out (FIFO). The former implies that tokens become available for output transitions immediately upon arrival just like in conventional QPN models. The latter implies that tokens become available for output transitions in the order of their arrival, i.e., a token can leave the place/depository only after all tokens that have arrived before it have left, hence the term FIFO. Coiming back to the problem above with the way thread contention is modeled, we now change place *G* to use the FIFO departure discipline. This ensures that subtransaction tokens waiting at place *G* are scheduled for processing in the order in which they arrive. After this change, the model passes the validation tests and can be used for performance prediction.

Performance Prediction

We consider two additional heavy load scenarios with an increased number of concurrent dealer clients leading to thread contention in the load balancer. The workload intensity parameters for the two scenarios are shown in Table 9.

Parameter	Heavy Load Sc. 3	Heavy Load Sc. 4
Browse Clients	300	270
Purchase Clients	30	90
Manage Clients	30	60
Planned Lines	120	120
Dealer Think Time	5 sec	5 sec
Mfg Think Time	10 sec	10 sec

Table 9. Workload intensity parameters for heavy load scenarios with thread contention

The first scenario has a total of 360 concurrent dealer clients, the second 420. Table 10 compares the model predictions for the first scenario in two configurations with 8 application servers and 15 and 30 load balancer threads, respectively. In addition to response times, throughput and utilization, the average length of the load balancer thread queue (N_{LBTQ}) is considered. As evident from the results, the model predictions are very close to the measurements and even for response times the modeling error does not exceed 16.4%. The results for the second scenario look very similar. The CPU utilization of the WebLogic servers and the database server increase to 63% and 52%, respectively, leading to slightly higher response times and lower throughput. The modeling error does

not exceed 15.2%. For lack of space, we do not include the detailed results. Repeating the analysis for a number of variations of the model input parameters led to results of similar accuracy.

	Heavy Load Sc. 3 with 15 Thr.			Heavy Load Sc. 3 with 30 Thr.		
METRIC	Model (99% c.i.)	Msrd.	Error	Model (99% c.i.)	Msrd.	Error
X_B	28.602 (+/- 1.001)	27.323	+4.7%	28.576 (+/- 1.000)	27.205	+5.0%
X_P	4.480 (+/- 0.157)	4.220	+6.2%	4.487 (+/- 0.157)	4.213	+6.5%
X_M	4.497 (+/- 0.159)	4.387	+2.5%	4.528 (+/- 0.158)	4.485	+1.0%
X_W	10.771 (+/- 0.377)	10.660	+1.0%	10.810 (+/- 0.378)	10.800	+0.1%
X_L	0.448 (+/- 0.016)	0.410	+9.3%	0.440 (+/- 0.015)	0.446	+0.1%
R_B	5494ms (+/- 40.8ms)	5740ms	-4.3%	5519ms (+/- 39.5ms)	5805ms	-4.9%
R_P	1673ms (+/- 16.3ms)	1977ms	-15.4%	1672ms (+/- 16.0ms)	2001ms	-16.4%
R_M	1683ms (+/- 17.3ms)	1779ms	-5.4%	1676ms (+/- 17.2ms)	1801ms	-6.9%
R_W	1124ms (+/- 2.2ms)	1158ms	-2.9%	1124ms (+/- 2.3ms)	1143ms	-1.7%
U_{LB}	99.9% (+/- 0.1)	93.0%	+7.5%	99.9% (+/- 0.1)	100.0%	0.0%
U_{AS}	57.8% (+/- 0.3)	57.8%	0.0%	57.8% (+/- 0.3)	58.0%	-0.3%
U_{DB}	41.5% (+/- 0.2)	44.0%	-5.7%	41.6% (+/- 0.2)	44.0%	-5.5%
N_{LBTQ}	146 (+/- 5.1)	161	-9.3%	131 (+/- 4.6)	146	-10.3%

Table 10. Analysis results for heavy load scenario 3 with 15 and 30 load balancer threads and 8 app. server nodes

4.10 Step 7: Analyze results and address modeling objectives

We can now use the results from the performance analysis to address the goals established in Section 4.3. By means of the developed QPN model, we were able to predict the performance of the system under peak operating conditions with 6 WebLogic servers. It turned out that using the original load balancer, six WebLogic servers were insufficient to guarantee average response times of business transactions below half a second. Upgrading the load balancer with a slightly faster CPU led to the CPU utilization of the load balancer dropping by a good 20 percent. As a result, the response times of dealer transactions improved by 14 to 26 percent, meeting the "half a second" requirement. However, increasing the workload intensity beyond peak conditions revealed that the load balancer was a bottleneck resource, preventing us to scale the system by adding additional WebLogic servers (see Figure 12).

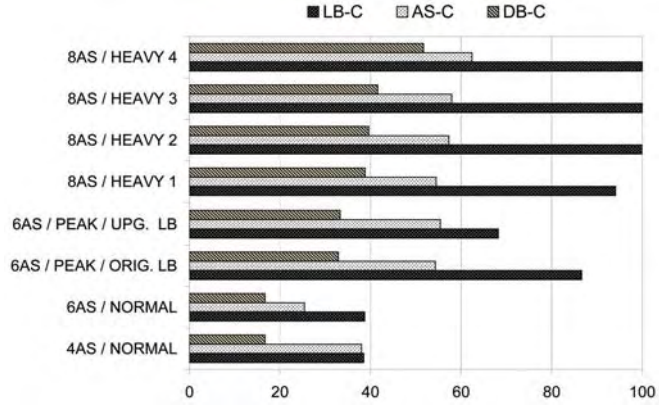


Fig. 12. Predicted server CPU utilization in considered scenarios.

Therefore, in light of the expected workload growth, the company should either replace the load balancer machine with a faster one or consider using a more efficient load balancing method. After this is done, the performance analysis should be repeated with the new load balancer to make sure that there are no other system bottlenecks. It should also be ensured that the load balancer is configured with enough threads to prevent thread contention.

5. Concluding remarks

In this chapter, we showed how QPNs can be exploited for the modeling and performance analysis of distributed systems. We presented a practical performance modeling methodology which helps to construct models of distributed systems that accurately reflect their performance and scalability characteristics. We started by introducing QPNs and discussing the state space explosion problem which is a major hurdle to their practical use. We then showed how the problem can be approached by exploiting discrete event simulation for model analysis. We presented SimQPN - a tool and methodology for simulating QPN models and analyzing the output data from simulation runs.

In the second part of the chapter, we presented a case study of a realistic distributed system, in which we showed how QPN models can be exploited as a powerful performance prediction tool in the software engineering process. The case study was used as an example in order to introduce a practical methodology for performance modeling and analysis of distributed systems. A deployment of the industry-standard SPECjAppServer2004 benchmark was studied, a large and complex application designed to be representative of today's distributed enterprise systems. It was shown in a step-by-step fashion how to build a detailed QPN model of the system, validate it, and then use it to evaluate the system performance and scalability. In addition to CPU and I/O contention, it was demonstrated how some complex aspects of system behavior such as composite transactions, software contention and asynchronous processing can be modeled. The developed QPN model was analyzed for a number of different deployment configurations and workload scenarios. The models demonstrated much better scalability and predictive power than what was achieved in our previous work. Even for the largest and most complex scenarios, the modeling error for transaction response time did not exceed 20.6% and was much lower for transaction throughput and resource utilization.

Taking advantage of the modeling power of QPNs, our methodology provides the following important benefits:

1. QPN models allow the integration of hardware and software aspects of system behavior and lend themselves very well to modeling distributed component-based systems.
2. In addition to hardware contention and scheduling strategies, using QPNs one can easily model software contention, simultaneous resource possession, synchronization, blocking and asynchronous processing.
3. By restricting ourselves to QPN models, we can exploit the knowledge of their structure and behavior for fast and efficient simulation using SimQPN. This enables us to analyze models of realistic size and complexity.
4. QPNs can be used to combine qualitative and quantitative system analysis. A number of efficient techniques from Petri net theory can be exploited to verify some

important qualitative properties of QPNs. The latter not only help to gain insight into the behavior of the system, but are also essential preconditions for a successful quantitative analysis [Bause, 1993].

5. Last but not least, QPN models have an intuitive graphical representation that facilitates model development.

To support the modeling and analysis of systems using QPNs, we have developed QPME (Queueing Petrinet Modeling Environment) [Kounev *et al.*, 2006]. QPME provides a user-friendly graphical interface enabling the user to quickly and easily construct QPN models. Model analysis is performed using SimQPN. Being implemented as an Eclipse application, QPME runs on all operating systems officially supported by the Eclipse platform. QPME provides a robust and powerful tool for performance analysis making it possible to exploit the modeling power and expressiveness of QPNs to their full potential.

6. Reference

Alexopoulos and Goldsman, 2004, C. Alexopoulos and D. Goldsman. To Batch Or Not To Batch. *ACM Transactions on Modeling and Computer Simulation,* 14(1):76-114, January 2004.

Alexopoulos and Seila, 2001, C. Alexopoulos and A. Seila. Output Data Analysis for Simulations. In *Proceedings of the 2001 Winter Simulation Conference, Arlington, VA, USA, December 9-12,* 2001.

Bause and Kritzinger, 2002, R Bause and R Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory.* Vieweg Verlag, second edition, 2002.

Bause *et al,* 1994, R Bause, P. Buchholz, and P. Kemper. Hierarchically Combined Queueing Petri Nets. In *Proceedings of the llth International Conference on Analysis and Optimization of Systems, Discrete Event Systems, Sophie-Antipolis (France),* 1994.

Bause, 1993, R Bause. Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, Toulouse, France, October 19-22,*1993.

Chien, 1994, C. Chien. Batch Size Selection for the Batch Means Method. In *Proceedings of the 1994 Winter Simulation Conference, Late Buena Vista, FL, USA, December 11-14,*1994.

Gaeta, 1996, Rossano Gaeta. Efficient Discrete-Event Simulation of Colored Petri Nets. *IEEE Transactions on Software Engineering,* 22(9), September 1996.

Heidelberger and Welch, 1983, P. Heidelberger and P. D. Welch. Simulation Run Length Control in the Presence of an Initial Transient. *Operations Research,* 31:1109-1145,1983.

Hogg and Craig, 1995, R. V. Hogg and A. R Craig. *Introduction to Mathematical Statistics.* Prentice-Hall, Upper Saddle River, New Jersey, 5th edition, 1995.

Jensen, 1981, K. Jensen. *Coloured Petri Nets and the Invariant Method.* Mathematical Foundations on Computer Science, Lecture Notes in Computer Science 118:327-338,1981.

Kounev and Buchmann, 2003, S. Kounev and A. Buchmann. Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS2003, Austin, Texas, USA, March 20-22,* 2003.

Kounev and Buchmann, 2006, S. Kounev and A. Buchmann. SimQPN - a tool and methodology for analyzing queueing Petri net models by means of

simulation. *Performance Evaluation,* 63(4-5):364-394, May 2006.

Kounev *et al,* 2006, S. Kounev, C. Dutz, and A. Buchmann. QPME - Queueing Petri Net Modeling Environment. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of SysTems (QEST-2006), Riverside, CA, September 11-14,*2006.

Kounev, 2006, S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Transactions on Software Engineering,* 32(7):486-502, July 2006.

Law and Kelton, 1982, A. Law and W. D. Kelton. Confidence Intervals for Steady-State Simulations, II: A Survey of Sequential Procedures. *Management Science,* 28(5)550-562, 1982.

Law and Kelton, 2000, Averill Law and David W. Kelton. *Simulation Modeling and Analysis.* Me Graw Hill Companies, Inc., third edition, 2000.

Menascé and Almeida, 1998, D. Menasce and V. Almeida. *Capacity Planning for Web Performance: Metrics, Models and Methods.* Prentice Hall, Upper Saddle River, NJ, 1998.

Menascé and Almeida, 2000, D. Menasce and V. Almeida. *Scaling for E-Business - Technologies, Models, Performance and Capacity Planning.* Prentice Hall, Upper Saddle River, NJ, 2000.

Menascé *et al,* 1994, Daniel A. Menasce, Virgilio A. E Almeida, and Larry W. Dowdy. *Capacity Planning and Performance Modeling - from Mainframes to Client-Server Systems.* Prentice Hall, Englewood Cliffs, NG, 1994.

Menascé *et al,* 1999, D. Menasce, V. Almeida, R. Fonseca, and M. Mendes. A Methodology for Workload Characterization of E-commerce Sites. In *Proceedings of the 1st ACM conference on Electronic commerce, Denver, Colorado, United States,* pages 119-128, November 1999.

Menascé *et al,* 2004, Daniel A. Menasce, Virgilio A. F. Almeida, and Lawrence W. Dowdy. *Performance by Design.* Prentice Hall, 2004.

Miller, 1974, R. G. Miller. The Jackknife: A Review. *Biometrika,* 61:1-15,1974.

Mortensen, 2001, Kjeld H. Mortensen. Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator. In *Proceedings of the 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, August 29-31,* 2001.

Pawlikowski *et al,* 1998, K. Pawlikowski, D. Mcnickle, and G. Ewing. Coverage of Confidence Intervals in Sequential Steady-State Simulation. *Journal of Simulation Practice and Theory,* 6(3):255-267,1998.

Pawlikowski, 1990, K. Pawlikowski. Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions. *ACM Computing Surveys,* 22(2):123-170,1990.

SPEC, 2004, SPEC. SPECjAppServer2004 Documentation. Specifications, April 2004.

Steiger *et al,* 2005, N. Steiger, E. Lada, J. Wilson, J. Joines, C. Alexopoulos, and D. Goldsman. ASAP3: a batch means procedure for steady-state simulation analysis. *ACM Transactions on Modeling and Computer Simulation,* 15(l):39-73, 2005.

Trivedi, 2002, K. S. Trivedi. Probability and Statistics with Reliability, Queuing and Computer Science Applications. John Wiley & Sons, Inc., second edition, 2002.

9

Model Checking of Time Petri Nets

Hanifa Boucheneb and Rachid Hadjidj
Department of Computer Engineering, École Polytechnique de Montréal
Canada

1. Introduction

To model time constraints of real time systems various time extensions are proposed, in the literature, for Petri nets. Among these extensions, time Petri nets model (TPN model) is considered to be a good compromise between modeling power and verification complexity. Time Petri nets are a simple yet powerful formalism useful to model and verify concurrent systems with time constraints (real time systems). In time Petri nets, a firing interval is associated with each transition specifying the minimum and maximum times it must be maintained enabled, before its firing. Its firing takes no time but may lead to another marking. Time Petri nets are then able to model time constraints even if the exact delays or durations of events are not known. So, time Petri nets are appropriate to specify time constraints of real time systems which are often specified by giving worst case boundaries.
This chapter reviews the well known techniques, proposed in the literature, to model check real time systems described by means of time Petri nets.
Model checking are very attractive and automatic verification techniques of systems (Clarke et al., 1999). They are applied by representing the behavior of a system as a *finite state transition system (state space)*, specifying properties of interest in a temporal logic (*LTL, CTL, CTL*, MITL, TCTL*) and finally exploring the state space to determine whether they hold or not. To use model checking techniques with timed models, an extra effort is required to abstract their generally infinite state spaces. Abstraction techniques aim to construct by removing some irrelevant details, a finite contraction of the state space of the model, which preserves properties of interest. For best performances, the contraction should also be the smallest possible and computed with minor resources too (time and space). Several abstractions, which preserve different kinds of properties, have been proposed in the literature for time Petri nets. The preserved properties can be verified using standard model checking techniques on the abstractions (Alur & Dill, 1990); (Penczek & Polrola, 2004); (Tripakis & Yovine, 2001).
This chapter has 6 sections, including introduction and conclusion sections. Section 2 introduces the time Petri nets model and its semantics. Section 3 presents the syntaxes and semantics of temporal logics *LTL, CTL, CTL** and *TCTL*. Section 4 is devoted to the TPN state space abstractions. In this section, abstractions proposed in the literature are presented, compared and discussed from state characterization, agglomeration criteria, preserved properties, size and computing time points of view. Section 5 considers a subclass of *TCTL* temporal logics and proposes an efficient on-the-fly model checking.

2. Time Petri nets

2.1 Definition of the TPN

A TPN is a Petri net with time intervals attached to its transitions. Formally, a TPN is a tuple $\aleph = (P, T, Pre, Post, m_0, Is)$ where

- P and T are finite sets of places and transitions such that $P \cap T = \varnothing$,
- Pre and $Post$ are the backward and the forward incidence functions: $P \times T \to N$[1],
- m_0 is the initial marking: $P \to N$,
- Is[2]: $T \to Q^+ \times (Q^+ \cup \{\infty\})$ associates with each transition t an interval called the *static firing interval* of t.

Let Int be an interval, $\downarrow Int$ and $\uparrow Int$ denote respectively its lower and upper bounds.
Let M be the set of all markings of a TPN, $m \in M$ a marking.
A transition t is said to be *enabled* in m, iff all tokens required for its firing are present in m, i.e.: $\forall p \in P,\ m(p) \geq Pre(p,t)$. We denote by $En(m)$ the set of all transitions enabled in m.
Two transitions t and t' of $En(m)$ are in conflict for m iff $(\exists p \in P,\ m(p) < Pre(p,t) + Pre(p,t'))$.
If m results from firing some transition t from another marking, $New(m,t)$ denotes the set of all transitions newly enabled in m, i.e.:

$$New(m,t) = \{t' \in En(m) \mid t' = t \vee (\exists p \in P,\ m(p) - Post(p,t) < Pre(p,t'))\}.$$

Note that only one instance of each transition is supposed to be active at the same time. A transition which remains enabled after firing one of its instance is considered newly enabled.

2.2 The semantics of the TPN

There are two known definitions of the TPN state. The first one, based on clocks, associates with each transition t of the model a *clock* to measure the time elapsed since t became enabled most recently (Yoneda & Ryuba, 1998); (Boucheneb & Hadjidj, 2004). The TPN *clock state* is a pair $s=(m,v)$, where m is a marking and v is a clock valuation function, $v: En(m) \to R^+$. The initial clock state of the TPN is $s_0 = (m_0,v_0)$, where m_0 is the initial marking and $v_0(t) = 0$, for all t in $En(m_0)$. The TPN state evolves either by time progression or by firing transitions. When a transition t becomes enabled, its clock is initialized to zero. The value of this clock increases synchronously with time until t is fired or disabled by the firing of another transition. t can fire, if the value of its clock is inside its static firing interval $Is(t)$. It must be fired immediately, without any additional delay, when the clock reaches $\uparrow Is(t)$. Its firing takes no time but may lead to another marking. Formally, let $\theta \in R^+$ be a nonnegative reel number, t a transition of T, $s=(m,v)$ and $s'=(m',v')$ two clock states of a TPN.

We write $s \xrightarrow{\theta} s'$ iff state s' is reachable from state s after a time progression of θ time units, i.e.: $m=m'$, $\forall t \in En(m),\ v(t)+\theta \leq \uparrow Is(t)$ and $\forall t \in En(m),\ v'(t)=v(t)+\theta$. s' is also denoted $s+\theta$.

We write $s \xrightarrow{t} s'$ iff state s' is immediately reachable from state s by firing transition t, i.e.: $\forall p \in P,\ m'(p)=m(p)-Pre(p,t)+Post(p,t)$, $t \in En(m)$, $v(t) \geq \downarrow Is(t)$ and $\forall t' \in En(m')$, $v(t')=$ if $t' \in New(m',t)$ then 0 else $v(t')$.

---

[1] N is the set of nonnegative integers.
[2] Q^+ is the set of nonnegative rational numbers.

The second characterization, based on intervals, defines the TPN state as a marking and a function which associates with each enabled transition a firing interval (Berthomieu & Vernadat, 2003). The TPN interval state is a couple $s=(m,I)$, where m is a marking and $I: En(m) \rightarrow Q^+ \times (Q^+ \cup \{\infty\})$ is an interval function. The initial interval state of the TPN is $s_0 = (m_0, I_0)$, where m_0 is the initial marking and $I_0(t) = Is(t)$, for all t in $En(m_0)$. When a transition t becomes enabled, its firing interval is set to its static firing interval $Is(t)$. The lower and upper bounds of this interval decrease synchronously with time, until t is fired or disabled by another firing. t can fire, if the lower bound of its firing interval reaches *0*, but must be fired, without any additional delay, if the upper bound of its firing interval reaches *0*. Its firing takes no time but may lead to another marking. Formally, let $\theta \in R^+$ be a nonnegative reel number, t a transition of T, $s=(m,I)$ and $s=(m',I')$ two interval states of a TPN.

We write $s \xrightarrow{\theta} s'$ iff state s' is reachable from state s after a time progression of θ time units, i.e.: $m=m'$ and $\forall t \in En(m)$, $\theta \leq \uparrow I(t) \wedge I'(t)=[max(0, \downarrow I(t)- \theta), \uparrow I(t)- \theta]$. s' is also denoted $s+\theta$.

We write $s \xrightarrow{t} s'$ iff state s' is immediately reachable from state s by firing transition t, i.e.: $\forall p \in P$, $m'(p)=m(p)-Pre(p,t)+Post(p,t)$, $t \in En(m)$, $\downarrow I(t)=0$, and $\forall t' \in En(m')$, $I(t')=$ if $t' \in New(m',t)$ then $Is(t')$ else $I(t')$.

The semantics of the TPN can be defined using either the clock state or interval state characterization. In both cases, the TPN *state space* is defined as a structure $(S, \rightarrow, s_0)$, where s_0 is the initial clock or interval state of the model, and $S = \{s \mid s_0 \xrightarrow{*} s\}$ is the set of reachable states ($\xrightarrow{*}$ is the reflexive and transitive closure of the relation $\rightarrow$ defined above).

Let s and s' be two TPN states, $\theta \in R^+$ and $t \in T$. As a shorthand, we write $s \xrightarrow{\theta t} s'$ iff $s \xrightarrow{\theta} s''$ and $s'' \xrightarrow{t} s'$ for some state s''. We write $s \overset{t}{\mapsto} s'$ iff $\exists \theta \in R^+, s \xrightarrow{\theta t} s'$.

An execution path in the TPN state space, starting from a state $s \in S$, is a maximal sequence $\rho = s^0 \xrightarrow{\theta_0 t_0} s^1 \xrightarrow{\theta_1 t_1} \ldots\ldots$, such that $s^0=s$. We denote by $\pi(s)$ the set of all execution paths starting from state s. $\pi(s_0)$ is therefore the sets of all execution paths of the TPN. The total elapsed time during an execution path ρ, denoted $time(\rho)$, is the sum $\sum_{i \geq 0} \theta_i$.

An infinite execution path is *diverging* if $time(\rho)=\infty$, otherwise it is said to be *zeno*. A TPN model is said to be non zeno if all its execution paths are not zeno. Zenoness is a pathological situation which suggests that infinity of actions may take place in a finite amount of time.

The TPN state space defines the *branching semantics* of the TPN model, whereas $\pi(s_0)$ defines its *linear semantics*. The graph of its execution paths, called *concrete state space*, is the structure $(\Sigma, \mapsto, s_0)$ where s_0 is the initial state of the model, $\Sigma = \{s \mid s_0 \overset{*}{\mapsto} s\}$ is the set of reachable concrete states of the TPN model, and $\overset{*}{\mapsto}$ is the reflexive and transitive closure of $\mapsto$.

3. Temporal logics for time Petri nets

Properties of timed systems are usually specified using temporal logics (Penczek & Polrola, 2004). We consider here CTL* (computation tree logic star) and a subclass of TCTL (timed

computation tree logic). Since our goal is to reason about temporal properties of time Petri nets, an atomic proposition is a proposition on a marking.
Let M be the set of reachable markings of a TPN model and PV the set of propositions on M, i.e., $\{\phi \mid \phi: M \to \{true, false\}\}$.

3.1 CTL* and its semantics

CTL^* is a temporal logic which allows to specify both linear and branching properties. The syntax of CTL^* is given by the following grammar:

$$\varphi_s \equiv false \mid \phi \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \forall\varphi_p \mid \exists\varphi_p$$
$$\varphi_p \equiv \varphi_s \mid \neg\varphi_p \mid \varphi_p \wedge \varphi_p \mid X\varphi_p \mid \varphi_p U \varphi_p$$

In the grammar, $\phi \in PV$ stands for an atomic proposition. φ_s and φ_p define respectively formulas that express properties of states and execution paths. $\forall$ ("for all paths") and $\exists$ ("there exists a path") are path quantifiers, whereas U ("until") and X ("next") are temporal operators (path operators). The sublogics CTL and LTL are defined as follows:

- In *CTL* formulas, every occurrence of a path operator is immediately preceded by a path quantifier: $\varphi \equiv false \mid \phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall(\varphi\ U\varphi\ ) \mid \exists(\varphi\ U\varphi\ ) \mid \forall X\varphi\ \mid \exists X\varphi$
- In *LTL*, formulas are supposed to be in the form $\forall\varphi_p$ where state subformulas of φ_p are propositions: $\varphi\ \equiv false \mid \phi \mid \neg\varphi\ \mid \varphi\ \wedge\varphi\ \mid \varphi\ U\varphi\ \mid X\varphi$

To ease CTL^* formulas writing, some abbreviations are used: $F\ \varphi = (true\ U\ \varphi)$, $\exists\varphi = \neg\forall\neg\varphi$ and $G\ \varphi = \neg F \neg\varphi$.
CTL^* formulas are interpreted on states and execution paths of a model $M = (S, V)$, where $S=(S, \to, s_0)$ is a transition system and $V: S \to 2^{PV}$ is a valuation function which associates with each state the subset of atomic propositions it satisfies.
Let $s \in S$ be a state of S, $\pi(s)$ the set of all execution paths starting from s, and $\rho = s^0 \xrightarrow{\theta_0 t_0} s^1 \xrightarrow{\theta_1 t_1} s^2 \xrightarrow{\theta_2 t_2} s^3 \ldots$ an execution path with ρ^i its suffix starting from s^i. The formal semantics of CTL^* is given by the satisfaction relation $\models$ defined as follows (the expression $M, s \models \varphi$ is read: "s satisfies property φ in the model M"):

- $\mathbf{M}, s \not\models false$,
- $\mathbf{M}, s \models \phi$ iff $\phi \in V(s)$,
- $\mathbf{M}, x \models \neg\varphi$ iff $\mathbf{M}, x \not\models \varphi$ for $x \in \{s, \rho\}$,
- $\mathbf{M}, x \models \varphi \wedge \psi$ iff $\mathbf{M}, x \models \varphi$ and $\mathbf{M}, x \models \psi$ for $x \in \{s, \rho\}$,
- $\mathbf{M}, s \models \forall\varphi$ iff $\forall\rho \in \pi(s), \mathbf{M}, \rho \models \varphi$,
- $\mathbf{M}, s \models \exists\varphi$ iff $\exists\rho \in \pi(s), \mathbf{M}, \rho \models \varphi$,
- $\mathbf{M}, \rho \models \varphi$ iff $\mathbf{M}, s^0 \models \varphi$, for a state formula φ,
- $\mathbf{M}, \rho \models X\varphi$ iff $\mathbf{M}, \rho^1 \models \varphi$,
- $\mathbf{M}, \rho \models \varphi\ U\ \psi$ iff $\exists j \geq 0, \mathbf{M}, \rho^j \models \psi$ and $\forall 0 \leq i < j, \mathbf{M}, \rho^i \models \varphi$.

We say that M satisfies φ, written $M \models \varphi$, iff $M, s_0 \models \varphi$. For instance $M, s_0 \models \forall(\varphi\ U\ \psi)$, iff for any execution path starting from s_0, φ is true in s_0 and the following states, until a state that satisfies ψ is reached.

To be able to specify explicitly time constraints of some important real-time properties such as, for example, the bounded response property, timed versions have been proposed for these logics (MITL, TCTL). Among these logics, we consider here a subclass of TCTL logic for an on-the-fly model checking.

3.2 TCTL and its semantics

TCTL is a timed extension of CTL (computation tree logic) where a time interval is associated with each temporal operator. The syntax of TCTL formulas is defined by the following grammar (in the grammar, $\phi \in PR$ and index I is an interval of $Q^+ \times (Q^+ \cup \{\infty\})$:

$$\varphi \equiv \mathit{false} \mid \phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall(\varphi \, U_I \, \varphi) \mid \exists(\varphi \, U_I \, \varphi)$$

TCTL formulas are also interpreted on states and execution paths of a model M =(S,V). To interpret a TCTL formula on an execution path, we introduce the notion of dense execution path. Let s∈S be a state of S, π(s) the set of all execution paths starting from s, and $\rho = s^0 \xrightarrow{\theta_0 t_0} s^1 \xrightarrow{\theta_1 t_1} \cdots\cdots$ an execution path of s. The dense path of the execution path ρ is the mapping $\hat{\rho} : R^+ \to S$ defined by: $\hat{\rho}(r) = s^i + \delta$ s.t. $r = \sum_{j=0}^{i-1} \theta_j + \delta$, i≥0 and $0 \le \delta \le \theta_i$.
The formal semantics of TCTL is given by the satisfaction relation = defined as follows:

- $\mathbf{M}, s \not\models \text{false}$,
- $\mathbf{M}, s \models \phi$ iff $\phi \in V(s)$,
- $\mathbf{M}, s \models \neg\varphi$ iff $\mathbf{M}, s \not\models \varphi$,
- $\mathbf{M}, s \models \varphi \wedge \psi$ iff $\mathbf{M}, s \models \varphi$ and $\mathbf{M}, s \models \psi$,
- $\mathbf{M}, s \models \forall(\varphi \, U_I \, \psi)$ iff $\forall \rho \in \pi(s), \exists r \in I, \mathbf{M}, \hat{\rho}(r) \models \psi$ and $\forall 0 \le r' < r, \mathbf{M}, \hat{\rho}(r') \models \varphi$.
- $\mathbf{M}, s \models \exists(\varphi \, U_I \, \psi)$ iff $\exists \rho \in \pi(s), \exists r \in I, \mathbf{M}, \hat{\rho}(r) \models \psi$ and $\forall 0 \le r' < r, \mathbf{M}, \hat{\rho}(r') \models \varphi$.

The *TPN* model is said to satisfy a *TCTL* formula φ iff $\mathbf{M}, s_0 \models \varphi$. When interval *I* is omitted, its value is *[0, ∞]* by default. Our timed temporal logic, we call $TCTL_{TPN}$, is defined as follows:

$$TCTL_{TPN} \equiv \forall(\varphi m \, U_I \, \varphi m) \mid \exists(\varphi m \, U_I \, \varphi m) \mid \varphi m \rightsquigarrow_{I_r} \varphi m \mid \exists G_I \, \varphi m \mid \forall G_I \, \varphi m \mid \exists F_I \, \varphi m \mid \forall F_I \, \varphi m$$

$$\varphi m ::= \varphi m \wedge \varphi m \mid \varphi m \vee \varphi m \mid \neg \varphi m \mid \phi \mid \mathit{false}$$

ϕ is a proposition on markings (i.e., $\phi \in PR$). *I* is a time interval. I_r is a time interval which starts from *0*. Formula $\phi_1 \rightsquigarrow_{I_r} \phi_2$ is a shorthand for *TCTL* formula $\forall G(\phi_1 \Rightarrow \forall F_{I_r} \phi_2)$ which expresses a bounded response property.
Several efficient model checking techniques were developed in the literature for *LTL, CTL, CTL*, MITL* and *TCTL,* using timed Büchi automata, fix point techniques, or hesitant alternating automata (Penczek & Polrola, 2004); (Tripakis et al., 2005); (Tripakis et al., 2001); (Visser & Barringer, 2000). To apply these techniques to time Petri nets, we must construct a finite abstraction for its generally infinite state space which preserves properties of interest.

4. TPN state space abstractions

Abstraction techniques aim to construct by removing some irrelevant details, a finite contraction of the state space of the model, which preserves properties of interest (markings, linear or branching properties). The preserved properties are then verified on the contraction using the classical model checking techniques. The challenge is to construct, with less resources (time and space), a much coarser abstraction preserving properties of interest.

4.1 Abstract state space

An abstract state space of the TPN model is defined as a structure $AS=(A, \Rightarrow, \alpha_0)$ where (Boucheneb & Hadjidj, 2006):

- A is a cover of S or Σ[3]. Each element α of A, called abstract state, is an agglomeration of some states sharing the same marking.
- α_0 is the initial abstract state class of AS, such that $s_0 \in \alpha_0$, and
- $\Rightarrow \subseteq A \times T \times A$ is the successor relation that satisfies condition EE, i.e.:

  i. $\forall(\alpha,t,\alpha') \in \Rightarrow,\ \exists s \in \alpha, \exists s' \in \alpha', s \overset{t}{\mapsto} s'$ and

  ii. $\forall(s,t,s') \in \mapsto,\ \forall \alpha \in A$ s.t. $s \in \alpha, \exists \alpha' \in A, (s' \in \alpha' \wedge \alpha \overset{t}{\Rightarrow} \alpha')$

The first part of condition EE prevents the connection of two abstract states with no connected states. The second one ensures that all sequences of transitions in the state space are represented in the abstraction.

Note that there are some differences between condition EE presented here and those given in (Berthomieu & Vernadat, 2003) and (Penczek & Polrola, 2004):

- EE in (Berthomieu & Vernadat, 2003):

$$\forall(\alpha,t,\alpha') \in A \times T \times A, \quad [(\exists s \in \alpha, \exists s' \in \alpha', s \overset{t}{\mapsto} s') \Leftrightarrow \alpha \overset{t}{\Rightarrow} \alpha']$$

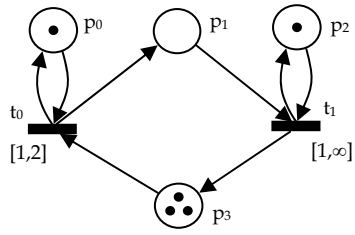
- EE in (Penczek & Polrola, 2004):

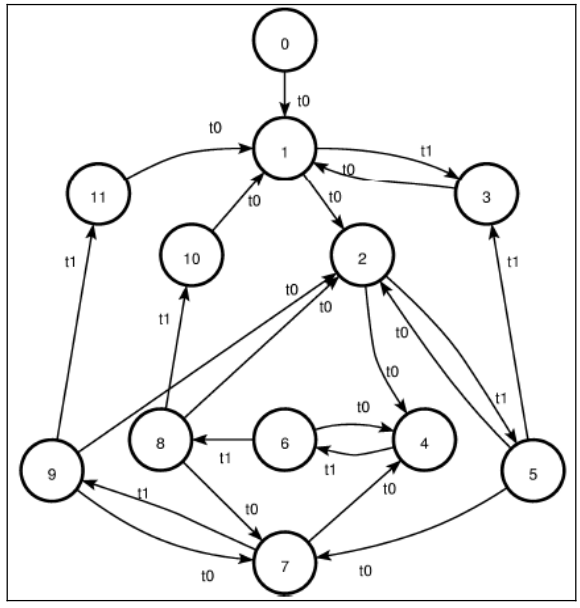
$$\forall(\alpha,t,\alpha') \in A \times T \times A, \ [(\exists s \in \alpha, \exists s' \in \alpha', s \overset{t}{\mapsto} s') \Rightarrow \alpha \overset{t}{\Rightarrow} \alpha']$$

Theses conditions impose to connect each two abstract states α and α' whenever some state of the first one has a successor in the second one. However, TPN abstractions proposed in the literature do not obey this rule, while they are still valid. As an example, consider the TPN with its Strong State Class Graph SSCG[4] shown in figure 1. Inequalities associated with each abstract state characterize the clock domains of all states agglomerated in the abstract state. The abstract state α_3 has a state which has a successor by t_0 in α_5, but no transition by t_0 exists from α_3 to α_5. In fact, the transition from α_3 to α_1 by t_0 ensures that some state in α_3 has as successor by t_0 the unique state of α_1. However, there is no transition from α_3 to α_5 by t_0, while the unique state of α_1 belongs also to α_5. This situation contradicts conditions EE given in (Berthomieu & Vernadat 2003) and (Penczek & Polrola, 2004).

[3] A cover of a set X is a collection of sets whose union contains X.

[4] The SSCG is a TPN abstraction proposed in (Berthomieu & Vernadat, 2003).





α_0: $p_0+p_2+3p_3$ $t_0=0$	α_1: $p_0+p_1+p_2+2p_3$ $t_0=t_1=0$	α_2: $p_0+2p_1+p_2+p_3$ $0\le t_0\le 0 \wedge 1\le t_1$	α_3: $p_0+p_2+3p_3$ $1\le t_0\le 2$
α_4: $p_0+3p_1+p_2$ $1\le t_1$	α_5: $p_0+p_1+p_2+2p_3$ $0\le t_0\le 2 \wedge t_1=0$	α_6: $p_0+2p_1+p_2+p_3$ $t_0=t_1=0$	α_7: $p_0+2p_1+p_2+p_3$ $t_0=0 \wedge 0\le t_1<1$
α_8: $p_0+p_1+p_2+2p_3$ $1\le t_0\le 2 \wedge t_1=0$	α_7: $p_0+p_1+p_2+2p_3$ $0<t_0\le 2 \wedge t_1=0$	α_7: $p_0+p_2+3p_3$ $t_0=2$	α_7: $p_0+p_2+3p_3$ $1<t_0\le 2$

Fig. 1. A TPN model and its SSCG.

The relation $\Rightarrow$ may satisfy other additional conditions such as (see figure 2):

$$EA:\ \forall(\alpha,t,\alpha')\in\Rightarrow,\ \ \forall s'\in\alpha',\exists s\in\alpha, s\overset{t}{\longmapsto}s',$$

$$AE:\ \forall(\alpha,t,\alpha')\in\Rightarrow,\ \ \forall s\in\alpha,\exists s'\in\alpha', s\overset{t}{\longmapsto}s'$$

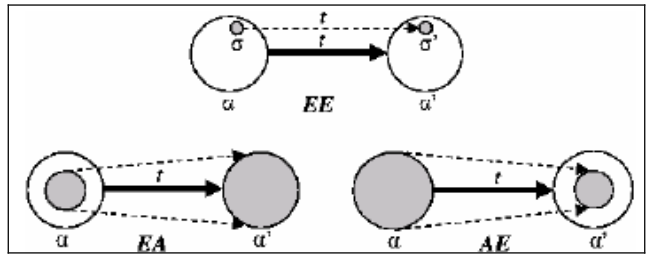


Fig. 2. Conditions *EE, EA* and *AE*.

A state class space which satisfies condition *AE* is called an *atomic state class graph*. An abstract state which satisfies condition *AE* for each outgoing edge is said to be atomic
The theorem below establishes a relation between conditions *AE*, *EA* and properties of the model preserved in the abstraction.

Theorem (Boucheneb & Hadjidj, 2006): Let $AS=(A,\Rightarrow,\alpha_0)$ be an abstraction of a TPN. Then:
1. If (*AS* satisfies condition *EA* and $\alpha_0=\{s_0\}$) then *AS* preserves *LTL* properties of the TPN,
2. If *AS* satisfies condition *AE* then it preserves *CTL** properties of the TPN.

4.2 Abstract states

We can find, in the literature, several state space abstractions for the TPN model: the state class graph SCG (Berthomieu & Vernadat, 2003), the zone based graph ZBG (Gardey & Roux, 2003), the geometric region graph GRG (Yoneda & Ryuba, 1998), the strong state class graph SSCG (Berthomieu & Vernadat, 2003) and the atomic state class graphs ASCGs (Boucheneb & Hadjidj, 2006); (Berthomieu & Vernadat, 2003); (Yoneda & Ryuba, 1998). These abstractions may differ mainly in the characterization of abstract states (interval states (Berthomieu & Vernadat, 2003), clock states (Boucheneb & Hadjidj, 2006) or firing dates (Yoneda & Ryuba, 1998), the agglomeration criteria of states, the kind of properties they preserve and their size.
In all these abstractions except the GRG, abstract states are defined as a couple $\alpha=(m,f)$, where m is a marking and f is a conjunction of atomic constraints of the form $x-y\prec c$, $-x\prec c$ or $x\prec c$, where $c\in Q\cup\{-\infty,\infty\}$, $\prec\in\{=,\leq,\geq,<,>\}$, and x, y are time variables. Each transition enabled in m is represented in f by a time variable, with the same name, representing either its delay or its clock ($Var(f)=En(m)$). All time variables are either clocks (clock abstract states) or delays (interval abstract states). Time variables are clocks in the SSCG, ZBG and ASCGs (clock state abstractions) but they are delays in the SCG (an interval state abstraction). Abstract states of the SCG, SSCG, ZBG and ASCGs are respectively called state classes, strong state classes, state zones and atomic state classes.
Abstract states of the GRG are triples (m,f,η), where m is the marking obtained by firing from the initial marking m_0 the sequence of transitions η and f is a set of atomic constraints on firing dates of transitions in m and their parents (transitions of η which made transitions of $En(m)$ enabled). This definition needs more time variables and constraints. It is therefore less interesting than those used in other abstractions. In addition, the relation of equivalence used in GRG involves large graphs and may induce infinite abstractions for some time Petri

nets with unbounded firing intervals (Berthomieu & Vernadat, 2003). Two abstract states are equivalent if they have the same marking, their enabled transitions have the same parents, and these parents could be fired at the same dates. For all these reasons, we do not consider here the abstract state definition of the GRG.

Though the same domain may be expressed by different conjunctions of atomic constraints, equivalent formulas have a unique form, called canonical form. Canonical forms make operations needed to compute and compare abstract states more simple. Let f be a conjunction of atomic constraints. The canonical form of f is:

$$f = \bigwedge_{x,y \in (Var(f) \cup \{o\})} x - y \prec_f^{x-y} Sup_f(x-y)$$

where *Var(f)* is the set of time variables of *f*, *o* represents the value *zero*, Sup_f *(x-y)* is the supremum of *x-y* in the domain of *f*, $\prec_f^{x-y}$ is either ≤ or <, depending respectively on whether *x-y* reaches its supremum in the domain of *f* or not. *Dom(f)* denotes the domain of *f*. By convention, we suppose that *Is(o)=[0,0]*.

The canonical form of *f* is usually represented by a DBM *B* (Daws et al., 1996) of order $|Var(f)|+1$, defined by:

$$\forall x, y \in (Var(f) \cup \{o\}), \mathrm{B}_{x\,y} = (Sup_f(x-y), \prec_f^{x-y})$$

An element of a DBM is called a bound. Operations like +, -, <, ≤, ≥, =, and min on bounds of DBMs are defined as usual: $\forall (c_1, \prec_1), (c_2, \prec_2) \in B, \forall \prec \in \{\leq, <, =, \geq\}$,

- $(c_1, \prec_1) \prec (c_2, \prec_2)$ iff ($c_1 \prec c_2 \wedge (c_1 = c_2 \Rightarrow \prec_1 \prec \prec_2)$); ("<" is less than operator "≤").
- $(c_1, \prec_1) + (c_2, \prec_2) = (c_1 + c_2, \min(\prec_1, \prec_2))$;
- $(c_1, \prec_1) - (c_2, \prec_2) = (c_1 - c_2, \min(\prec_1, \prec_2))$;
- $-(c_1, \prec_1) = (-c_1, \prec_1))$
- $\min((c_1, \prec_1), (c_2, \prec_2)) = if\ (c_1, \prec_1) \leq (c_2, \prec_2)\ then\ (c_1, \prec_1)\ else\ (c_2, \prec_2)$

The computation of canonical forms is based on the shortest path *Floyd-Warshall*'s algorithm and is considered as the most costly operation (cubic in the number of variables in *f*) (Berhmann et al., 2002). In (Boucheneb & Mullins, 2003) and (Boucheneb & Hadjidj, 2006), authors have shown how to compute, in $O(n^2)$, for respectively the SCG and the SSCG, the canonical form of each successor abstract state, n being the number of variables in the abstract state. An abstract state is said in canonical form iff its formula is in canonical form.

The convexity of abstract states is an important criterion to maintain their computation simple. The simplicity of the method is particularly guaranteed by the usage of DBMs. This data structure adapts well to all computation aspects involved in constructing abstractions, but fails to efficiently represent non convex domains (DBMs are not closed under set-union). To avoid this limitation, *Clock Difference Diagrams* (CDDs) (Larsen et al., 1999) seems to be a better alternative. CDDs allow to represent in a very concise way the union of convex domains. They are also closed under set-union, intersection and complementation. However, due to the lack of a known simple computing canonical form, CDDs fail to compete with DBMs when it comes to computing successors and predecessors of abstract

states. A detailed description of CDDs can be found in (Berhmann et al., 2002) and (Larsen et al., 1999), where the authors use this data structure to represent computed state zones in the list PASSED of the reachability algorithm, implemented in the tool UPPAAL. Yet, they still use DBMs to compute successors of abstract states. Note that abstract states within the list PASSED are handled using only two basic operations which are well supported by CDDs (set-union and inclusion).

4.3 Abstractions preserving linear properties

An abstraction is said to preserve linear properties if it has exactly the same firing sequences as its concrete state space. In abstractions preserving linear properties, we distinguish, in general, three levels of abstraction (see figure 3). In the first level, states reachable by time progression may be either represented (ZBG) or abstracted (SCG, GRG, SSCG). In the second level, states reachable by the same firing sequence independently of their firing times are agglomerated in the same node. In the third level, the agglomerated states are then considered modulo some relation of equivalence (firing domain of the SCG (Berthomieu & Vernadat, 2003), the approximations of the ZBG (Gardey & Roux, 2003) and the SSCG (Berthomieu & Vernadat, 2003)). These abstractions, except the GRG, are finite for all bounded time Petri nets. Indeed, for some bounded TPNs with unbounded static firing intervals, the GRG may be infinite. However, in (Pradubsuwun et al., 2005), authors used the approximation of timed automata to ensure the convergence of the construction of the GRG for bounded TPNs with unbounded static firing intervals.

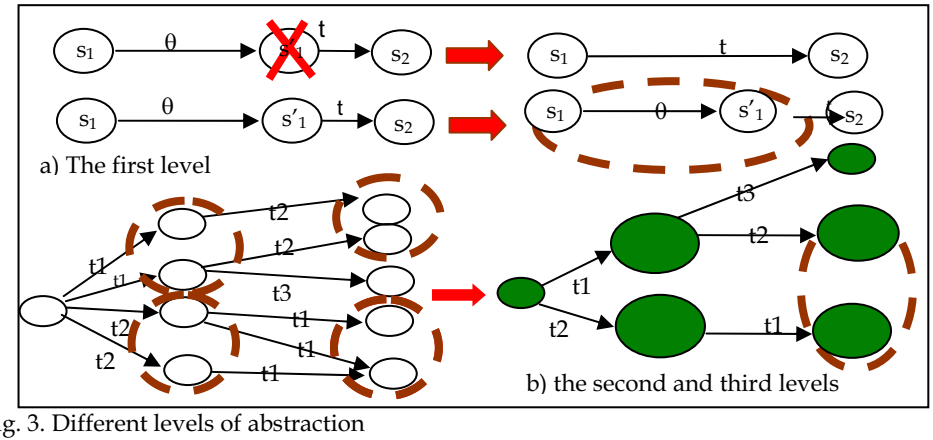


Fig. 3. Different levels of abstraction

4.3.1 Basic operations on abstract states

Let α be an abstract state and t a transition of T. We define the basic operations on α, used to construct abstractions preserving linear properties:

- $succ(\alpha,t) \overset{def}{=} \{s' \mid \exists s \in \alpha, s \xrightarrow{t} s'\}$ is the set of all states reachable from α by firing immediately transition t.

- $\vec{\alpha} \overset{def}{=} \{s' \mid \exists s \in \alpha, \exists \theta \in R^+, s \xrightarrow{\theta} s'\}$ contains α and all states reachable from α via some time progression.

Let us now show how to compute successor abstract states for clock abstract states and for interval abstract states.

Let $\alpha=(m,f)$ be a clock abstract state in canonical form, and t a transition of T:

$succ(\alpha,t)\neq\varnothing$ iff $t \in En(m)$ and $f \wedge \downarrow Is(t) \leq t$ is consistent. This means that there is at least a state in α such that t is firable from it (its clock reaches its static firing interval).

If $succ(\alpha,t)\neq\varnothing$ then $succ(\alpha,t) = (m',f')$ is computed in four steps:

1. $\forall p \in P$, $m'(p)=m(p)-Pre(p,t)+Post(p,t)$.
2. Initialize f' with $f \wedge \downarrow Is(t) \leq t$. This step eliminates from f states from which t is not immediately firable.
3. Put f' in canonical form and eliminate t and all transitions conflicting with t in m.
4. Add constraints $\bigwedge_{t' \in New(m',t)} t' = 0$ and put f' in canonical form (clocks of newly enabled transitions are set to 0).

- $\vec{\alpha} = (m, f')$ is computed in three steps:
  1. Initialize f' with f;
  2. Replace all constraints $t - o \leq c$ with $t - o \leq \uparrow Is(t)$. Clocks increase with time until reaching upper bounds of the static firing intervals of their transitions or their transitions are fired or disabled.
  3. Put f' in canonical form.

Let $\alpha=(m,f)$ be an interval abstract state in canonical form, and t a transition of T:

- $succ(\alpha,t)\neq\varnothing$ iff $t \in En(m)$ and $f \wedge t = 0$ is consistent. This means that there is at least a state in α such that t is firable from it (its delays is equal to 0).

  If $succ(\alpha,t)\neq\varnothing$ then $succ(\alpha,t) = (m',f')$ is computed in four steps:
  1. $\forall p \in P$, $m'(p)=m(p)-Pre(p,t)+Post(p,t)$.
  2. Initialize f' with $f \wedge t = 0$. This step eliminates from f states from which t is not immediately firable.
  3. Put f' in canonical form and eliminate t and all transitions conflicting with t in m.
  4. Add constraints $\bigwedge_{t' \in New(m',t)} \downarrow Is(t') \leq t' \leq \uparrow Is(t')$ and put f' in canonical form. The firing interval of each newly enabled transition is set to its static firing interval.

- $\vec{\alpha} = (m, f')$ is computed in three steps:
  1. Initialize f' with f;
  2. Replace each constraint $o - t \leq c$ with $o - t \leq 0$. Delays decrease with time until reaching 0 or their transitions are fired or disabled.
  3. Put f' in canonical form.

4.3.2 Approximation of clock abstract states

Let $\alpha=(m,f)$ be a clock abstract state in canonical form and $En_{=\infty}(m)=\{t \mid t \in En(m) \wedge \uparrow Is(t)=\infty\}$ the set of unbounded transitions enabled in m.

The SSCG approximation of α denoted $approx_{SSCG}(\alpha)$ produces a partition of α: $\{(m, f_e) \mid (e \subseteq En_{=\infty}(m) \vee e=\varnothing\}$, where f_e is a consistent formula characterizing states of (m, f) in which all transitions of e have not yet reached their minimal delays, while those of $En_{=\infty}(m)-e$ have reached or over-passed their minimal delays. f_e is computed in three steps:

1. Initialize f_e with: $f \wedge (\bigwedge_{t\in e} t <\downarrow Is(t)) \wedge (\bigwedge_{t'\in En_{=\infty}(m)-e} t' \geq \downarrow Is(t'))$;
2. Put f_e in canonical form and eliminate all variables in $En_{=\infty}(m)-e$;
3. Add the constraint $\bigwedge_{t'\in En_{=\infty}(m)-e} \downarrow Is(t') \leq t'$.

Steps (2) and (3) extend f_e with possibly non reachable states when replacing the domain of each variable t' in $En_{=\infty}(m)-e$ by $[\downarrow Is(t'),\infty]$. Nevertheless, these states correspond all to the same interval state (Berthomieu & Vernadat, 2003). Therefore, this operation preserves linear properties of the abstract state α.

Let k be the greatest finite bound appearing in the static firing intervals of the considered TPN. The ZBG approximation of α, proposed in (Gardey & Roux, 2003), denoted $approx_k(\alpha)$, is the abstract state (m,f') where f' is the canonical form of the formula computed from f as follows: For each $t \in En_{=\infty}(m)$, $x \in En(m)\cup\{o\}$,

1. Replace constraint $x - t \prec c$ with $x - t \leq -k$, if $c <- k$;
2. Remove constraint $t - x \prec c$ if $k < c$.

Step (1) replaces by k the lower bound of $t - x$ which exceeds k ($x - t \prec c <- k$ is equivalent to $k < - c \prec t - x$). Step (2) is equivalent to replace by ∞ the upper bound of $t - x$ which exceeds k. This operation extends f with possibly non reachable states but the added states do not alter linear properties of the abstract state α (Gardey & Roux, 2003). In (Boucheneb et al., 2006), authors proposed two other approximations for the ZBG, denoted respectively $approx_{kx}$ and $approx_{kx'}$ which lead to much compact graphs. They showed that α, $approx_{kx}(\alpha)$ and $approx_{kx'}(\alpha)$ have the same firing domain and then the same firing sequences.

$approx_{kx}(\alpha)$ is the abstract state (m,f') where f' is the canonical form of the formula computed from f as follows: For each $t \in En_{=\infty}(m)$, $x \in En(m)\cup\{o\}$,

1. Replace constraint $x - t \prec c$ with $x - t \leq -\downarrow Is(t)$, if $c \leq -\downarrow Is(t)$;
2. Remove constraint $t - x \prec c$, if $\downarrow Is(t) \prec c$.

$approx_{kx'}(\alpha)$ is the abstract state (m,f') where f' is the canonical form of the formula computed from f as follows: For $t, t' \in En(m)$,

1. Replace the constraint $o - t \prec c$ with $o - t \leq 0$, if $t \in En_{=\infty}(m)$;
2. Remove constraint $t' - t \prec c'$ if $t\in En_{=\infty}(m)$ or the constraint $o - t \prec c$ is s.t. $c' - \downarrow Is(t') \geq c$.

$approx_{kx'}$ has been integrated recently in the tool Romeo[5] in replacement of the one proposed in (Gardey & Roux, 2003). This approximation is referred in the sequel as $approx_{ZBG}$.

4.3.3 Construction of abstractions preserving linear properties

An abstraction preserving linear properties is generated progressively by computing the successors of the initial abstract states and those of each newly computed abstract state, until no more new abstract states are generated. All computed abstract states are considered

[5] *http://romeo.rts-software.org*

modulo some relation of equivalence. In table 1, we give the formal definition of the SCG, ZBG and SSCG from which the construction algorithms can be derived.

AS	SCG	ZBG	SSCG
Initial abstract state	(m_0,f_0) $f_0 = \underset{t\in En(m)}{\wedge} \downarrow Is(t) \le t \le \uparrow Is(t)$	$approx_{ZBG}(\overrightarrow{(m_0,f_0)})$ $f_0 = \underset{t\in En(m)}{\wedge} t = 0$	$approx_{SSCG}((m_0,f_0))$ $f_0 = \underset{t\in En(m)}{\wedge} t = 0$
$(\alpha,t,\alpha') \in \Rightarrow_{AS}$	$succ(\vec{\alpha},t) \neq \varnothing \wedge$ $\alpha' = succ(\vec{\alpha},t)$	$succ(\alpha,t) \neq \varnothing \wedge$ $\alpha' = approx_{ZBG}(\overrightarrow{succ(\alpha,t)})$	$succ(\vec{\alpha},t) \neq \varnothing \wedge$ $\alpha' \in approx_{SSCG}(succ(\vec{\alpha},t))$
A	$\{\alpha \mid \alpha_0 \Rightarrow_{SCG} \alpha\}$	$\{\alpha \mid approx_{ZBG}(\vec{\alpha}_0) \overset{*}{\Rightarrow}_{ZBG} \alpha\}$	$\{\alpha \mid \alpha_0 \overset{*}{\Rightarrow}_{SSCG} \alpha\}$

Table 1. Definition of SCG, ZBG and SSCG.

4.3.4 Interval state abstractions versus clock state abstractions

Clock based abstractions are less interesting than the SCG when only linear properties are of interest. They are in general larger, and their computation takes more time. The origin of these differences stems from the relationship between the two characterizations of states which can be stated as follows: Let *(m,v)* be a clock state. Its corresponding interval state is *(m,I)* s.t. $\forall t \in En(m)$, $I(t) = [max(0, \downarrow Is(t)-v(t)), \uparrow Is(t)-v(t)]$. Note that for any real value $u \geq \downarrow Is(t)$, if $\uparrow Is(t) = \infty$, $\uparrow Is(t) - u = \infty$ and $max(0, \downarrow Is(t)-u)=0$. This means that many clock states may map to the same interval state. In such a case, all these states will obviously exhibit the same future behaviour. The same remark extends also to interval abstract states and clock abstract states. As an example, consider the model shown in figure 4.a. The repetitive firing of transition t_0, from the initial abstract state, generates 2 strong state classes sc_1 and sc_2 (figure 4.c) which map to the state class c_1 (figure 4.b). Moreover, the number of strong state classes which map to c_1 depends and increases with the value of $\uparrow Is(t_1)$. For example, for $\uparrow Is(t_1) = 9$, we obtain 5 strong state classes which correspond to the state class c_1.

Moreover, abstractions based on clocks do not enjoy naturally the finiteness property for bounded TPNs with unbounded intervals as it is the case for abstractions based on intervals. The finiteness is enforced using an *approximation* operation on clock abstract states, which may involve some overhead computation. Another point which contributes to generate coarser abstractions concerns states reachable by time progression. We obtain coarser abstractions when we add to each abstract state all states reachable from it by time progression (relaxing abstract states). Indeed, two different abstract states may have the same relaxed abstract state. As an example, the two SCG state classes $\alpha_1=(m, 2 \le t \le 3)$ and $\alpha_2=(m, 1 \le t \le 3)$ are s.t. $\alpha_1 \neq \alpha_2$ and $\vec{\alpha}_1 = \vec{\alpha}_2 = (m, 0 \le t \le 3)$. To achieve more contractions, we define a relaxed version to the SCG, named relaxed state class graph (RSCG), as a structure $(A, \Rightarrow_{RSCG}, \vec{\alpha}_0)$ where:

1. $\alpha_0=(m_0,f_0)$ where m_0 is the initial marking and $f_0 = \underset{t\in En(m)}{\wedge} \downarrow Is(t) \le t \le \uparrow Is(t)$.
2. $\forall \alpha, \alpha', t, (\alpha,t,\alpha') \in \Rightarrow_{RSCG}$ iff $succ(\alpha,t) \neq \varnothing$ and $\alpha' = \overrightarrow{succ(\alpha,t)}$.
3. $A = \{ \alpha \mid \overrightarrow{\alpha_0} \Rightarrow_{RSCG} \alpha \}$.

However, abstractions based on intervals are not appropriate for constructing abstractions preserving branching properties (*ASCGs*). Indeed, this construction, based on splitting abstract states, is not possible on state classes (the union of intervals is irreversible) whereas it is possible on clock abstract states. Together, the mentioned remarks suggest that the

interval characterization of states is more appropriate to construct abstractions preserving linear properties but is not appropriate to construct abstractions preserving branching properties.

We have implemented and tested several abstractions. We report in table 2 sizes (nodes/edges) and computing times of the RSCG, SCG, SSCG and ZBG we obtained for the producer consumer model (figure 5) and the level crossing model (figure 6). The level crossing model *T(n)* is obtained by putting in parallel one copy of the controller model, *n* copies of the train model (with $m = n$) and one copy of the barrier model. Trains and the barrier are synchronized with the controller on transitions with the same names. The producer consumer model *P*(*n*) is the parallel composition of *n-1* copies of the model in figure 6.b with one copy of the model in figure 6.a while merging all places named P_1 in one single place. The obtained results confirm that the RSCG is in general smaller and faster to compute too.

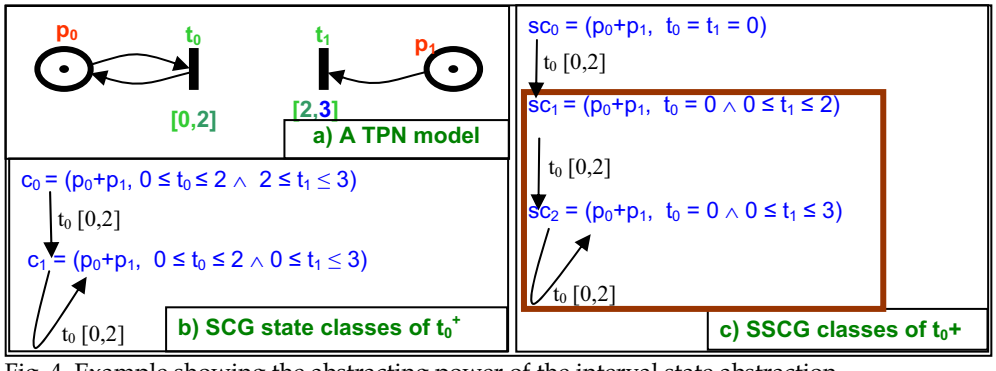


Fig. 4. Example showing the abstracting power of the interval state abstraction.

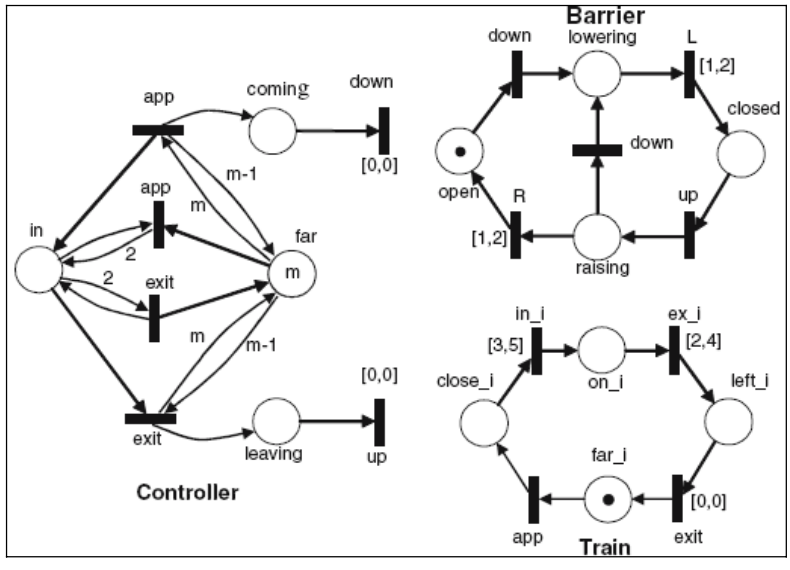


Fig. 5. The crossing level model

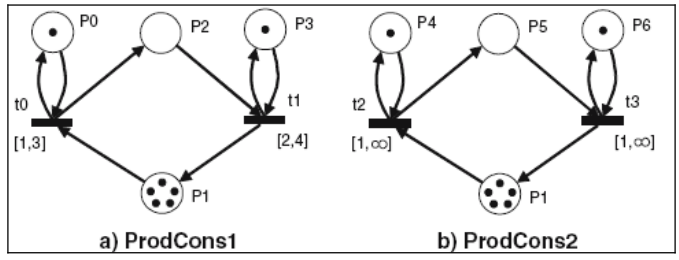


Fig. 6. The producer consumer model

TPN	RSCG	SCG	SSCG	ZBG ($approx_{kx'}$)	ZBG ($approx_{kx}$)
P(2) cpu(s)	593 / 1922 0.01	748 / 2460 0.02	7963 / 42566 0.73	593 / 1922 0.14	2941 / 9952 0.31
P(3) cpu(s)	3240 / 15200 0.12	4604 / 21891 0.30	122191 / 1111887 37.86	3240 / 15200 0.20	100060 / 385673 210.22
P(4) cpu(s)	9267 / 54977 0.73	14086 / 83375 1.76	?[6]	9504 / 56038 1.05	?
P(5) cpu(s)	20877 / 145037 2.01	31657 / 217423 5.67	?	20877 / 145037 13.06	?
T(2) cpu(s)	113 / 198 0	123 / 218 0	141 / 254 0	114 / 200 0	147 / 266 0
T(3) cpu(s)	2816 / 6941 0.07	3101 / 7754 0.09	5051 / 13019 0.5	2817 / 6944 0.18	5891 / 15383 0.54
T(4) cpu(s)	122289 / 391240 5.74	134501 / 436896 6.33	?	122290 / 391244 9.40	?

Table 2. Comparison of abstractions preserving linear properties

4.4 Abstractions preserving branching properties

Abstractions preserving branching properties (*CTL** properties) are built using a partition refinement technique in two steps (Paige & Tarjan, 1987). An abstraction, which does not necessarily preserve branching properties, is first built then refined in order to restore the condition *AE* (the resulting graph is atomic).

4.4.1 Refinement process

Let $AS = (A, \Rightarrow, \alpha_0)$ be an abstract state space of a TPN model, $\alpha=(m,f)$, $\alpha'=(m',f')$ two abstract states of A, t a transition of T s.t. $(\alpha, t, \alpha') \in \Rightarrow$ and $pred(\alpha', t, \alpha) \overset{def}{=} \{s \in \alpha \mid \exists s' \in \alpha', s \overset{t}{\mapsto} s'\}$. To verify the atomicity of α for the edge (α, t, α'), it suffices to verify that α is equal or included

[6] The computation has not completed after an hour of time, or aborted due to a lack of memory.

in $pred(\alpha', t, \alpha)$. In case α is not atomic, it is partitioned into a set of convex subclasses so as to isolate the predecessors of α' by t in α, from those which are not.
$Pred(\alpha', t, \alpha)=(m,f'')$ is computed in five steps:

1. Initialize f'' to $f' \wedge \bigwedge_{t' \in New(m',t)} t' = 0$,
2. Put f'' in canonical form and eliminate by substitution all transitions in $New(m', t)$,
3. Add constraints: $\downarrow Is(t) \leq t$, $\bigwedge_{t' \in En(m')} t' \leq \uparrow Is(t')$ and $\theta \geq 0$,
4. Replace each variable t by $t + \theta$, put f'' in canonical form then eliminate θ,
5. Add all constraints of f and put f'' in canonical form.

Knowing that the firing of transition t sets the clock of each newly enabled transition to zero, step (1) extracts from α' the subset of states where the clocks of newly enabled transitions are equal to zero. Step (3) adds the firing constraints of transition t. Step (4) goes back in time (each clock is decreased by θ time units). Finally, step (5) adds all constraints of class α.
Since the domain of the difference is not necessarily convex, we construct a partition of α - $Pred(\alpha', t, \alpha)$ such that all its parts are convex. Let $\alpha = (m, f)$ and $\alpha' = (m, f')$ be two abstract states such that $\alpha' \subseteq \alpha$. A partition of the complement of α' in α, denoted $Comp(\alpha, \alpha')$, is computed as follows:

```
Algorithm Comp(α = (m, f), α' = (m, f'))
{   Part:=∅;
    X:= f;
    For each atomic constraint g of f'
    {   if (X ∧ ¬g) is consistent then   Part := Part ∪ {(m, X ∧ ¬g);
        X:= (X ∧ f);
    }
    Return Part;
}
```

The refinement proceeds according to the following algorithm: After its splitting, α is replaced by its partition. Each subclass inherits all connections of α in accordance with condition *EE*. The refinement step is repeated until condition *AE* is established. The refinement process generates a finite graph iff the intermediate abstraction is finite (Berthomieu & Vernadat, 2003).

```
Algorithm Refine(AS)

{ Repeat {   For each α ∈ A such that α is not atomic for some transition α ⟹ α'
                 {   α'':=Pred(α', t, α);
                     Part: = Comp( α, α'');
                     Part:= Part ∪ {α''};
                     Replace α by Part in AS;
                 }
         } while (AS is not atomic)
}
```

4.4.2 Intermediate abstractions

The intermediate abstractions used in (Yoneda & Ryuba, 1998) (GRG) and (Berthomieu & Vernadat, 2003) (SSCG) preserve linear properties. However, these abstractions are in general large graphs with a high degree of state redundancy (the same state may appear in several abstract states). Experimental results showed that this redundancy induces the refinement procedure to waste time and space computing redundant abstract states. For instance, if an abstract state is included into another one, refining both abstract states may result in identical atomic abstract states. If both abstract states are replaced by the most including one, no pertinent information will be lost while refinement steps get reduced. To reduce state redundancy in abstraction preserving linear properties, we proposed to group together abstract states whenever one of them includes all the others (Boucheneb & Hadjidj, 2006) or their union is convex (Boucheneb & Hadjidj, 2004). When a set of abstract states are grouped, they are replaced by a new abstract state representing their union. All transitions between these abstract states become loops for their union. Ingoing and outgoing transitions of the grouped abstract states become respectively ingoing and outgoing of their union. If one of the grouped abstract states contains the initial abstract state, their union becomes the initial abstract state. The contraction may be performed either during or at the end of the construction. With these abstractions, we obtain an important reduction in refinement times and memory usage, resulting in graphs closer in size to the optimal (see table 3). Despite the simplicity of the used models, they allowed to illustrate some interesting features related to the computation pattern followed by the refinement procedure, depending on which abstraction is refined (see figure 7). If an inclusion or convex-combination abstraction is used, the refinement follows a linear pattern (i.e., the size of the graph grows linearly in time during its construction). When an abstraction preserving linear properties is refined, the size of the computed graph starts first to grow up to a *peek size* then decreases until an atomic state class space is obtained. In certain cases, the peek size grows out of control, leading to a state explosion.

The inclusion test is performed as follows: Let $\alpha=(m,f)$ and $\alpha'=(m,f')$ be two abstract states sharing the same marking and B, B' their DBMs in canonical form. (m,f) is included in (m,f') iff: $\forall x, y \in En(m) \cup \{o\}$, $B_{x\,y} \leq B'_{x\,y}$.

For the convex-combination, before explaining how to perform the grouping of abstract states, we first define what a *convex-hull* is. Let $\alpha=(m,f)$, $\alpha'=(m,f')$ be two abstract states sharing the same marking (see figure 8):

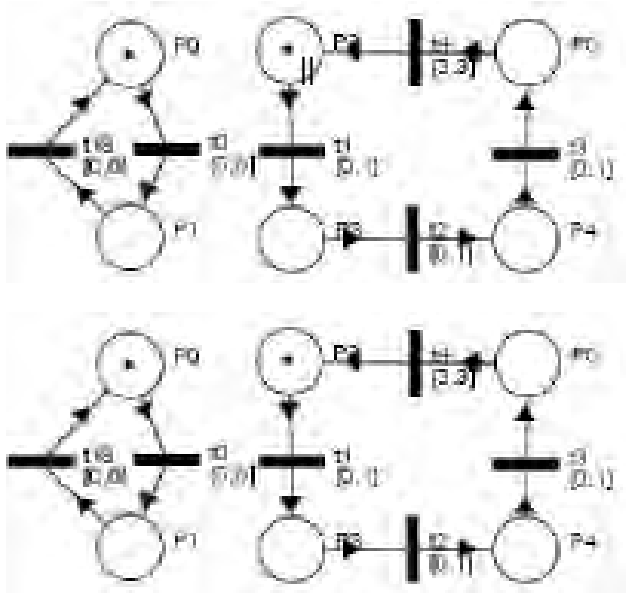
- The *convex-hull* of α and α', denoted $\hat{\alpha}_{(\alpha,\alpha')}$, is the abstract state $\alpha''=(m,f'')$ where:

$$f'' = \bigwedge_{x,y \in En(m)\cup\{o\}} x - y \prec^{x-y}_{f \vee f'} Sup_{f \vee f'}(x-y)$$

- Let $\alpha''=(m,f'')$ be the convex-hull of α and α'. $\alpha''=(m,f'')$ is the canonical form of the union of α and α' iff $(Dom(f'') - Dom(f)) \subseteq Dom(f')$.

The convex-combination test of two abstract states involves three operations: convex-hull, complement of a domain and a test of inclusion. Moreover, abstract states which may not combine two by two may combine three by three or more. Figure 9 illustrates some situations involving the convex combination of abstract states with two enabled transitions only. In case *a)*, abstract states α and α' are combined into the abstract state α''. Case *b)* shows two abstract states whose union is not convex and therefore cannot be grouped by convex combination. Case *c)* illustrates a situation where three abstract states α, α' and α'' cannot combine when taken two by two, but combine well in α'' if taken all together. Cases

d) and *e)* show other situations, where the grouping two by two is not possible, but becomes possible for other grouping.



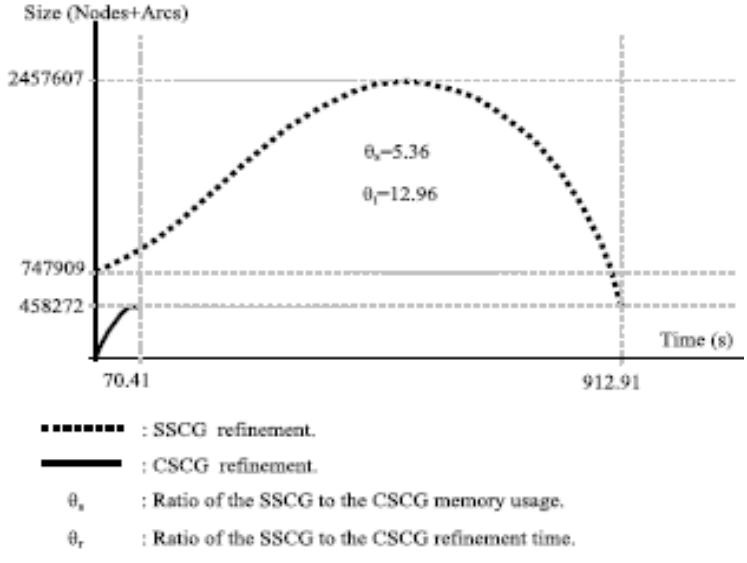


Fig. 7. A TPN model and the refinement patterns of its SSCG and CSCG[7]

[7] CSCG is a contraction by inclusion of the SSCG.

TPN	Refining SSCG	Refining CSCG	Refining CCSCG	Optimal
P(2) cpu(s)	2615/ 28263 8.42	2444 / 26358 1.15	2411 /26138 1.01	2334 / 25046 9.41
P(3) cpu(s)	?	31197 / 485960 40.18	30828 / 480987 35.62	28319 / 430875 3887.30
P(4) cpu(s)	?	151384 / 2887295 358.06	151384 / 2887295 358.06	?
T(2)	195 / 849 0.02	192 / 844 0.02	188 / 814 0.01	185 / 786 0.03
T(3)	6983 / 50044 5.00	6966 / 49802 2.11	6918 / 49025 1.49	6905 / 48749 60.88
T(4)	?	356940 / 3447624 288.21	356930 / 3447548 317.29	?

Table 3. Refining SSCG, CSCG and CCSCG.

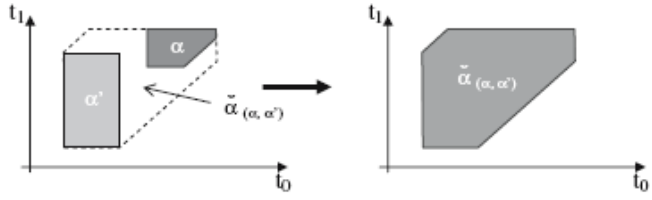


Fig. 8. Convex-hull of two abstract states

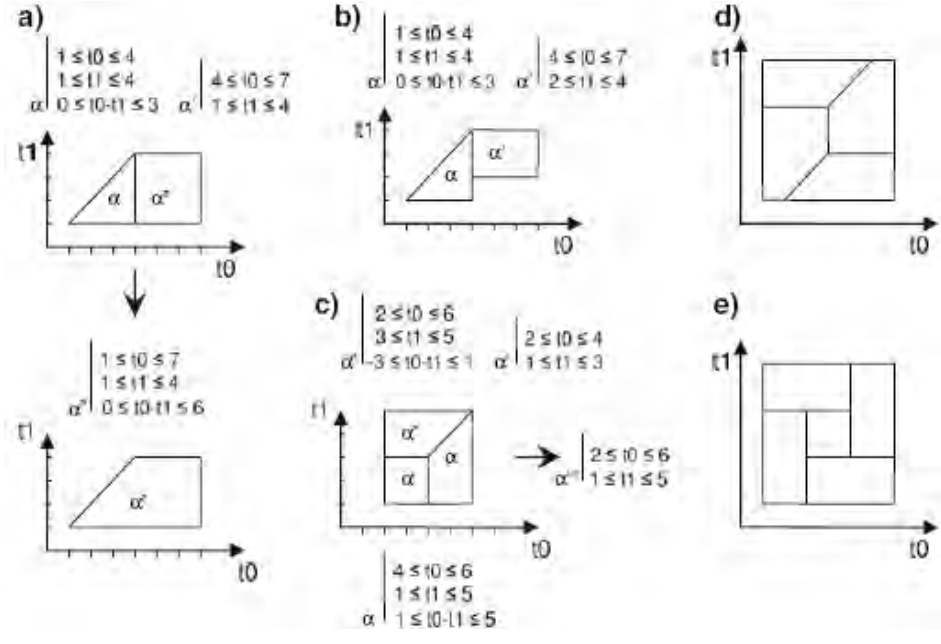


Fig. 9. Grouping abstract states by convex-combination.

To achieve a high degree of contraction, we need to test all possible combinations of abstract states sharing the same marking and having states in common. But this operation is computationally very expensive. Experimental results have shown that performing the test on abstract states two by two, results in very satisfactory contractions, in relatively short computing times too. Furthermore, when two abstract states are such that one is included into the other, their convex combination is simply the most including abstract state. So, before performing the convex combination test, we check first for inclusion in $O(n^2)$, where n is the number of transitions enabled in the shared marking of the two abstract states.
All CTL* model checking techniques can be applied directly on the atomic state class graphs to determine linear and branching properties of time Petri nets. All states within the same atomic abstract state have the same CTL* properties and are then considered as an indivisible unit.

5. Model checking timed properties of time Petri nets

To verify some timed properties, in (Toussaint, J. et al., 1997), authors used observers to express them in the form of TPNs and reduce them to reachability properties. However, properties on markings are quite difficult to express with observers. Other techniques define translation procedures from the TPN model into timed automata (Cassez & Roux, 2006); (Lime & Roux, 2003), in order to make use of available model checking techniques and tools (Penczek & Polrola, 2004); (Tripakis et al., 2005). Model checking is then performed on the resulting timed automata, with results interpreted back on the original TPN model. The translation into timed automata may be either structural (each transition is translated into a timed automata using the same pattern) (Cassez & Roux, 2006) or semantic (the state class graph of the TPN is first constructed and then translated into a timed automaton) (Lime & Roux, 2003). Such translations show that *CTL*, TCTL, LTL, MITL* model checking are decidable for bounded TPNs and that developed algorithms on timed automata may be extended to TPNs. Though effective, these techniques face the difficulty to interpret back and forth properties between the two models. In (Virbitskaite & Pokozy, 1999), authors proposed a method to model check TCTL properties of TPN. The method is based on the region graph method and is similar to the one proposed in (Alur & Dill, 1990) for timed automata. However, the region graph is known to be a theoretical method which is not applicable in practice because of its lack of efficiency.
To achieve the same goal, it is possible to adapt to the TPN, the method proposed in (Penczek & Polrola, 2004) and (Tripakis et al., 2005) for timed automata. The verification of a TCTL formula proceeds by adding a transition named t_s[8] to the *TPN*, translating the *TCTL* formula into some *CTL* formula, constructing an abstraction which preserves *CTL* properties of the completed TPN and then applying a *CTL* model checking technique. The transformation of *TCTL* formulas into *CTL* ones needs to extend *CTL* with atomic propositions of the form $t_s \in I$, and a particular next operator X_{ts} defined by: for each formula ψ and each state s' of the TPN, s' satisfies $X_{ts}\ \psi$ iff the state resulting by firing t_s satisfies ψ.

[8] This transition is used to deal with time constraints of the property to be verified. Its firing interval is $[0,\infty]$.

For example, the formula $\varphi = \forall(\varphi_1 \ U_I \ \varphi_I)$ is translated into the formula $\varphi' = X_{ts} \ (\forall (\varphi_1' \ U \ (\varphi_2' \wedge t_s \in I))$. The verification of φ' is performed using the classical *CTL* model checking technique by constructing an abstraction which preserves φ'. However, this method needs to compute the whole abstraction of the model before it is analyzed and then runs up against the state explosion problem. To attenuate the state explosion problem, on-the-fly model checking methods may be a good alternative, as they allow to verify a property during the construction of an abstraction preserving linear properties. The construction of the graph is stopped as soon as the truth value of the property is obtained. On-the-fly methods have proven to be very effective to model-check a subclass of *TCTL* on zone graphs of timed automata. So, they can be straightforward adapted to clock based abstractions of time Petri nets. However, TPN abstractions based on intervals are in general smaller and faster to compute than TPN abstractions based on clocks. So, applying on-the-fly methods on TPN abstractions based on intervals should give better performances. In this sense, in (Hadjidj & Boucheneb, 2006), we proposed, using the state class method (SCG), a forward on-the-fly model checking technique for a subclass of TCTL properties. The verification proceeds by augmenting the TPN model under analysis with a special TPN, called *Alarm* shown in figure 10, to allow the capture of relevant time events (reaching, over passing a time interval). A forward on-the-fly exploration combined with an abstraction by inclusion is then applied on the resulting TPN. In the sequel, we give algorithms to model check $TCTL_{TPN}$ properties.
Note that all following developments apply similarly to both the SCG and the RSCG. The SCG will be considered for explanations.
Let $\aleph$ be a TPN model and $\varphi=\phi_1 \rightsquigarrow_{[0,b]} \phi_2$. Model checking φ on $\aleph$ could be performed by analyzing each execution path of the TPN SCG, until the truth value of φ is established. The SCG is progressively constructed, depth first, while looking for the satisfaction of property ϕ_1. If ϕ_1 is satisfied at an abstract state α, ϕ_2 is looked for in each execution paths which starts from α (i.e., $\forall \rho \in \pi(\alpha)$). For each execution path $\rho \in \pi(\alpha)$, ϕ_2 is required to be satisfied at a state class α' such that the time separating α and α' is within the time interval *[0,b]*. If this is the case the verification of φ is restarted again from α', and so forth, until all state classes are explored. Otherwise, the exploration is stopped, and φ is declared invalid.

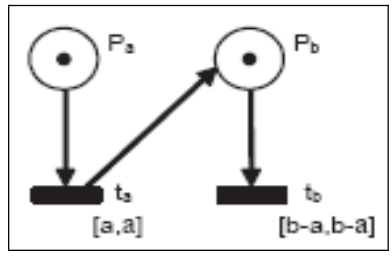


Fig. 10. The Alarm TPN

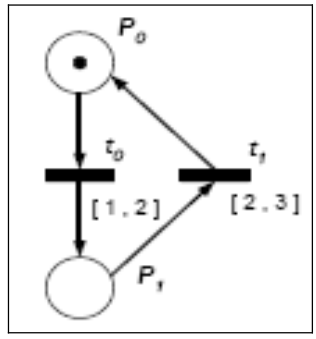


Fig. 11. cyclic TPN model

Some attention is required when dealing with transitions t_a and t_b. If transition t_a can be fired at exactly the same time as another transition t, and t is fired before t_a, φ might be declared wrongly false if the resulting state class satisfies ϕ_2. A similar situation might arise for transition t_b if it is fired before a transition t which can be fired at exactly the same time. To deal with these two special situations, we assign a *high firing priority* to transition t_a, so that it is fired before any other transition which can be fired at exactly the same time. At the contrary, we assign a *low firing priority* to t_b so that it is fired after any other transition which can be fired at exactly the same time. To cope with this priority concepts, we need to change the way we decide if a transition is firable or not, and the way the successor of a state class $\alpha=(m,f)$, by a transition t, is computed (i.e., operation *succ*). $succ_{AC}(\alpha,t)$ replaces $succ(\alpha,t)$ to check whether a transition is firable or not and compute successor state classes. What changes is the way the firing condition *fc* is computed:

1. If ($t \neq t_a$ and $t_a \in En(m)$) then $fc = f \wedge 0 = t < t_a$
2. If ($t=t_b$ and $t_b \in En(m)$) then $fc = f \wedge t = 0 \wedge (\bigwedge_{t' \in En(m)-\{t_b\}} t_b < t')$
3. If ($t=t_a$ or ($t_a \notin En(m)$ and $t \neq t_b$)) then $fc = f \wedge t = 0$.

In case t_a is enabled while we want to fire a different transition t (case 1), we need to make sure that t is fired ahead of time of t_a. In case t_b is enabled and is the one we want to fire (case 2), we need to make sure that t_b is the only transition that can be fired. The remaining cases are handled exactly as before.

$Succ_{AC}(\alpha,t) \neq \varnothing$ iff *fc* is consistent. If $succ_{AC}(\alpha,t) \neq \varnothing$ then $succ_{AC}(\alpha,t) = (m',f')$ is computed in four steps:

1. $\forall p \in P$, $m'(p)=m(p)-Pre(p,t)+Post(p,t)$.
2. Initialize f' with fc. This step eliminates from f states from which t is not immediately firable.
3. Put f' in canonical form and eliminate t and all transitions conflicting with t for m.
4. Add constraints $\bigwedge_{t' \in New(m',t)} \downarrow Is(t') \leq t' \leq \uparrow Is(t')$ and put f' in canonical form. The firing interval of each newly enabled transition is set to its static firing interval.

The verification of φ proceeds as follows: During the generation of the SCG of $\mathcal{N}||Alarm$, if ϕ_1 is satisfied in a state class $\alpha=(m,f)$, transition t_a is enabled in α to capture the event corresponding to the beginning of time interval I_r. t_a is enabled by changing the marking m in α such that place P_a would contain one token, and replacing f with $f \wedge t_a=a$. These two actions correspond to artificially putting a token in place P_a of $Alarm$. Since a=0 and transition t_a has the highest priority, it is fired before all others. When t_a is fired (which means that time has come to start looking for ϕ_2), t_b gets enabled in the resulting state class $\alpha=(m,f')$ to capture the event corresponding to the end of interval I_r. If t_b is fired during the exploration, φ is declared invalid and the exploration stops. If before firing t_b, ϕ_2 is satisfied in a state class $\alpha''=(m'',f'')$ transition t_b is disabled in α'' by changing the marking m'' such that place P_b would contain zero tokens, and eliminating variable t_b from f''. These two actions correspond to artificially removing the token in place P_b. After α'' is modified, φ is checked again starting from α''. Note that in this technique, the fact of knowing a state class and the transition that led to it, is sufficient to know which action to take[9]. This means that there is no need to keep track of execution paths during the exploration, and hence, the exploration strategy of the SCG (depth first, breadth first,..) is irrelevant. This in turn solves the problem of dealing with cycles and infinite execution paths for bounded TPN models.

Let $\alpha=(m,f)$ be a state class and t the transition that led to it. The different cases that might arise during the exploration are given in what follows:

1. The case where t_a, $t_b \notin En(m)$ and $t \notin \{t_a,t_b\}$ corresponds to a situation where we are looking for ϕ. In case ϕ_1 is satisfied in α, we enable t_a in α,
2. The case where $t_b \in En(m)$ corresponds to a situation where we are looking for ϕ_2. If ϕ_2 is satisfied in α then we disable t_b and get in a situation where we are looking for ϕ_1 (i.e., (1)).
3. The case where $t=t_b$ corresponds to a situation where interval I_r has expired while we are looking for ϕ_2. In this case, we stop the exploration and declare φ invalid.

Another problem may arise for zeno TPNs. Indeed, if the model is zeno and has a zeno execution path such that all its state classes satisfy ϕ_1 but its time is less that b. In this case, t_b will never get fired to signal the end of interval I_r, and the verification would conclude that the property is valid while it is not. To correct this problem, one solution consists in detecting zeno cycles during the verification, but not any zeno cycle. The zeno cycles of interest are only those which arise when transition t_a or t_b is enabled.

Algorithm modelCheck(φ)
*{ continue:=true; /*global variable */*
*valid:=true; /*global variable */*
COMPUTED:= $\emptyset$;
α_0 := (m_0,f_0);

---

[9] For uniformity reasons, we assume a fictitious transition t_ε as the transition which led to the initial state class.

α_0':= $checkStateClass_\varphi(\alpha_0, t_\varepsilon)$;
WAIT={α_0'};
while (continue)
{ remove α=(m,f) from WAIT;
for ($t \in En(m)$ s.t. $succ_{AC}(\vec{\alpha}, t) \neq \emptyset$) provided continue
{ $\alpha' := succ_{AC}(\vec{\alpha}, t)$;
If ($\varphi \neq \exists(\phi_1 \, U \, \phi_2)$ and ($t_a \in En(m)$ or $t_b \in En(m)$) and $\downarrow Is(t) = 0$) then Connect α to α';
$\alpha'' := checkStateClass_\varphi(\alpha', t)$;
if (continue $\wedge$ $\alpha'' \neq \emptyset$ $\wedge$ $\nexists \alpha_p \in$ COMPUTED s.t. $\alpha'' \subseteq \alpha_p$) then
{ for($\alpha_p \in$ COMPUTED s.t. $\alpha_p \subseteq \alpha''$) remove α_p from COMPUTED and from WAIT;
add α'' to COMPUTED and to WAIT;
}
}
}
If ($\varphi \neq \exists(\phi_1 \, U \, \phi_2)$ and COMPUTED has a cycle s.t. t_a or t_b is enabled in all its state classes) then
valid := false;
Return valid;
}

The on-the-fly $TCTL_{TPN}$ model checking of formula φ is based on the following exploration algorithm *modelCheck*(φ). This algorithm uses two lists: *WAIT* and *COMPUTED*, to manage state classes, and calls a polymorphic satisfaction function $checkStateClass_\varphi$ to check the validity of formula φ. *COMPUTED* contains all computed state classes, while *WAIT* contains state classes of *COMPUTED* which are not yet explored. The algorithm generates state classes by firing transitions. The initial state class is supposed to result from the firing of a fictive transition t_ε. Each time a state class α is generated as the result of firing a transition t, α and t are supplied to $checkStateClass_\varphi$ to perform actions and take decisions. In general, $checkStateClass_\varphi$ enables or disables transitions t_a and t_b in α. It also takes decisions, and record them in two global boolean variables continue and valid, to guide the exploration process. Finally, it returns either α after modification or ∅ in case α needs to be no more explored (i.e., ignored). The exploration continues only if continue is *true*. *valid* is used to record the truth value of φ. After $checkStateClass_\varphi$ is called, the state class α' it returns is inserted in the list WAIT only if it is not included in a previously computed state class. Otherwise, α' is inserted in the list *WAIT*, while all state classes of the list *COMPUTED* which are included into α' are deleted from both *COMPUTED* and *WAIT*. This strategy, used also in the tool *UPPAAL* (Behrmann et al., 2002), attenuates considerably the state explosion problem. So instead of exploring both α and α', exploring α' is sufficient. Operation $checkStateClass_\varphi$ takes as parameters: a state class, and the transition that led to it. Three different implementations of $checkStateClass_\varphi$ are required for the three principal forms of φ, i.e., $\phi_1 \rightsquigarrow_{Ir} \phi_2$, $\forall(\phi_1 \, U_I \, \phi_2)$ and $\exists(\phi_1 \, U_I \, \phi_2)$, with $I=[a,b]$ and $I_r=[0,b]$ (bound b can be either finite or infinite). All of these implementations handle four mutually exclusive cases corresponding to four types of state classes that can be encountered on an execution path.

The first implementation corresponds to property $\varphi=\phi_1 \rightsquigarrow_{Ir} \phi_2$. The first case it handles corresponds to a state class not reached by the firing t_a nor t_b, and neither of them is enabled

in it. The remaining cases correspond respectively to: a state class where transition t_b is enabled and a state class reached by the firing of transition t_b.

Algorithm $checkStateClass_{\phi 1 \rightsquigarrow Ir\ \phi 2}(\alpha=(m,f),t)$
{ *if* ($t_a, t_b \notin En(m) \wedge t \notin \{t_a, t_b\}$) *then*
if($\phi_1(m)$) *then* *enable* t_a *in* α;
if($t_b \in En(m) \wedge \phi_2(m)$) *then* *disable* t_b *in* α;
if ($t=t_b$) *then* { *valid=false ; continue=false; }*
Return α;
}

The second implementation corresponds to property $\varphi=\forall(\phi_1 U_I \phi_2)$. In it first case, this implementation looks for the initial state class only. The remaining cases are similar to those of the first implementation, but different actions are taken for each one of them. Intuitively the verification of property $\varphi=\forall(\phi_1 U_I \phi_2)$ checks if proposition ϕ_1 is true in the initial state class and all state classes following it, until t_a fires. From the moment t_a is fired, the verifier checks for the satisfaction of either ϕ_1 or ϕ_2, until ϕ_2 is true or t_b is fired. If ϕ_2 becomes true in a state class α, α is no more explored. In case t_b is fired, the exploration is stopped and the property is declared invalid.

Algorithm $checkStateClass_{\forall(\phi 1\ UI\ \phi 2)}(\alpha=(m,f),t)$
{ *if*($t=t_a$) *then*
{ *if* ($\phi_1(m)$) *then* *enable* t_a *in* α;
else *if*($\neg\ \phi_2(m) \vee a>0$) *then* { *valid=false; continue=false; }*
else { *valid=true; continue=false; }*
}
if ($t_a \in En(m) \wedge \neg\ \phi_1(m)$) *then* { *valid=false; continue=false; }*
if ($t_b \in En(m)$) *then*
if ($\neg\ \phi_2(m)$) *then*
{ *if* ($\neg\ \phi_1(m)$) *then* { *valid=false; continue=false; }*
} *else* *Return* $\varnothing$;
if ($t=t_b$) *then* { *valid=false; continue=false; }*
Return α;
}

The implementation of $checkStateClass_{\exists(\phi 1\ UI\ \phi 2)}$ corresponds to property $\varphi=\exists(\phi_1 U_I \phi_2)$. It handles four similar cases as the previous implementation, but different actions are taken. For instance, this implementation initializes variable valid to false as soon as the initial state class is entered, and stops the exploration of a state class α if it does not comply with the semantics of φ. It also aborts the exploration as soon as a satisfactory execution path is found.

To illustrate our verification approach, we consider the simple TPN model shown in figure 11, we call cyclic. The $TCTL_{TPN}$ property we verify is $\varphi= \phi_1 \rightsquigarrow_{[0,3]} \phi_2$, with proposition $\phi_1(m)= (m(P_0)=0)$ and proposition $\phi_2(m)= (m(P_1)=1)$. For simplicity reasons, we selected a cyclic TPN model with a single execution path, for which property φ is trivially valid.

The verification process of φ starts first by constructing the TPN model cyclic||Alarm, such that *a=0* and *b=3*, then runs according to the following steps:

1. Compute the initial state class of *cyclic||Alarm*: $\alpha_0 = (P_0, 1 \leq t_0 \leq 2)$.
2. Check if ϕ_1 is valid in α_0: ϕ_1 is not valid in α_0.
3. Fire t_2 from α_0 and put the result in α_1: $\alpha_1 = (P_1, 2 \leq t_1 \leq 3)$.
4. Check if ϕ_1 is valid in α_1: ϕ_1 is valid in α_1.
5. Enable t_a in α_1: α_1 becomes $((P_1+P_a, 2 \leq t_1 \leq 3 \wedge t_a=0)$.
6. Fire t_a from α_1 and put result in α_2: $\alpha_2 = (P_1+P_b, 2 \leq t_1 \leq 3 \wedge t_b=3)$.
7. Check if ϕ_2 is satisfied in α_2: ϕ_2 is not satisfied in α_2.
8. Fire t_1 from α_2 and put the result in α_3: $\alpha_3 = (P_0+P_b, 1 \leq t_0 \leq 2 \wedge 0 \leq t_b \leq 1)$.
9. Check if ϕ_2 is satisfied in α_3: ϕ_2 is satisfied in α_3.
10. Disables t_b in α_3: α_3 becomes $(P_0, 1 \leq t_0 \leq 2)$.
11. Declare φ valid since α_3 has already been explored ($\alpha_3=\alpha_0$).

We have implemented and tested this approach on the level classical model. The properties we considered are:

12. The gate is never open whenever a train is crossing: $\varphi_1 = \forall G \neg(open \wedge \bigvee_{1 \leq i \leq n} on_i)$.
13. If a train approaches, the gate closes in less than 2 time units: $\varphi_2 = coming \rightsquigarrow_{[0,2]} closed$.
14. The level crossing model is deadlock free: $\varphi_3 = \forall G\ (En(m) \neq \varnothing)$.

Table 3 reports results obtained for model checking the selected properties using our approach, applied on the SCG. Each result is given in terms of the final size of the list COMPUTED and the total number of explored state classes, followed by the exploration time. The second column recalls the size and computing time of the ASCGs. All properties have been successfully tested valid.

TPN	ASCG	φ_1	φ_2	φ_3
T(2) cpu(s)	188 / 814 0.01	38 / 116 0	41 / 91 0	38 / 116 0
T(3) cpu(s)	6918 / 49025 1.49	173 / 790 0	182 / 646 0.01	173 / 790 0.01
T(4) cpu(s)	356930 / 3447548 317.29	1176 / 7162 0.12	1194 / 6073 0.1	1176 / 7162 0.12
T(5) cpu(s)	?	10973 / 81370 2.37	11008 / 71152 2.04	10973/81370 2.30
T(6) cpu(s)	?	128116/1103250 110.81	128184/986939 100.92	128116/1103250 111.18

Table 4. Comparison of ASCGs with our on-the-fly method

6. Conclusion

In this chapter, we presented and discussed model checking techniques of time Petri nets. We pointed out some strategies which allow to make model checking techniques more efficient. For model checking LTL properties, we proposed a contraction for the state class

graph (SCG), called RSCG, which is both smaller and faster to compute than other abstractions. For CTL* model checking, we showed that refining abstractions contracted by inclusion or convex-combination allow to improve significantly the refinement process. For all tested models, the refinement follows a linear pattern when an inclusion or convex-combination abstraction is used. When an abstraction preserving linear properties is refined, the size of the computed graph starts first to grow up to a *peek size* then decreases until an atomic state class space is obtained. Finally, to attenuate the state explosion problem of model checking techniques, we considered a subclass of TCTL and proposed an on-the-fly method for the RSCG and SCG. On-the-fly methods have proven to be very effective to model-check a subclass of *TCTL* of timed automata.

7. References

Alur, R. & Dill, D. (1990) . Automata for modelling real-time systems, Proceedings of 17ème ICALP, LNCS 443, pp. 322–335. Springer-Verlag, 1990.

Behrmann, G. ; Bengtsson, J.; David, A.; Larsen, K. G.; Pettersson, P. & Yi, W. (2002). UPPAAL Implementation Secrets, Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 2469, pp. 3–22. Springer-Verlag, 2002.

Berthomieu, B. & Vernadat, F. (2003). State class constructions for branching analysis of time Petri nets, In Proceedings of TACAS 2003, LNCS 2619, pp. 442–457. Springer-Verlag. 2003.

Boucheneb, H.; Gardey, G. & Roux. O. H. (2006). TCTL model checking of time Petri nets. Technical Report IRCCyN number RI2006-14, 2006.

Boucheneb, H. & Hadjidj, R. (2006). CTL* model checking for time Petri nets, Theoretical Computer Science journal, vol. 353(1-3)(1-3), pp. 208-227, 2006.

Boucheneb, H. & Hadjidj, R. (2004). Towards optimal CTL* model checking of Time Petri Nets, Proceedings of the International Workshop on Discrete Event Systems (WODES). Reims-France, 2004.

Boucheneb, H. & Mullins, J. (2003). Analyse de réseaux de Petri temporels. Calculs des classes en $O(n^2)$ et des temps de chemin en $O(m \times n)$, Technique et Science Informatiques, vol. 22, no. 4, 2003.

Bucci, G. & Vicario, E. (1995). Compositional validation of time-critical systems using communicating Time Petri nets, IEEE transactions on software engineering, vol. 21, no. 12. pp. 969–992 December 1995.

Cassez, F. & Roux, O. H. (2006). Structural translation from time Petri nets to timed automata, Journal of Systems and Software, 79(10), pp. 1456-1468, 2006

Clarke, E. M.; Grumberg, O. & Peled, D. (1999). Model Checking, MIT Press, Cambridge, MA. 1999.

Daws, C.; Olivero, A.; Tripakis, S. & Yovine, S. (1996). The tool Kronos, In Hybrid Systems III, Verification and Control, LNCS 1066, pp. 208–219, Springer-verlag, 1996.

Gardey, G. & Roux, O. H. Using zone graph method for computing the state space of a time Petri net, In Formal Modeling and Analysis of Timed Systems (FORMATS), LNCS 2791, pp 246-259, Springer-Verlag, Marseille, France, September 2003.

Hadjidj, R. & Boucheneb, H. (2006). On-the-fly TCTL model checking for time Petri nets using the state class method, In Proceedings of the 6th International Conference on

Application of Concurrency to System Design (ACSD), IEEE Computer Society Press, 2006.

Hadjidj, R. & Boucheneb, H. (2005). Much compact Time Petri Net state class spaces useful to restore CTL* properties, In Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD), IEEE Computer Society Press, 2005

Henzinger, T. A.; Ho, P-H. & Wong-Toi, H. (1997). HyTech : A Model Checker for Hybrid Systems, Software Tools for Technology Transfer 1, 1997.

Larsen, K.G.; Weise, C.; Yi, W. & Pearson, J. (1999) Clock difference diagrams. Nordic J. Comput. **26**(3), pp. 271–298 (1999).

Lime, D. & Roux, O. H. (2003). State class timed automaton of a time Petri net, In Proceedings of the 10th Int. Workshop on Petri Nets and Performance Models (PNPM). IEEE Comp. Soc. Press, 2003.

Paige, R. & Tarjan, R. (1987). Three partition refinement algorithms. SIAM, J. Comput. **16**(6), pp. 973–989 (1987).

Penczek, W. & Polrola, A. (2004). Specification and Model Checking of Temporal Properties in Time Petri Nets and Timed Automata, In Proceedings of ICATPN'01, pp. 37–76, 2004.

Pettersson, P. (1999). Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice, Ph.D. thesis, Uppsala University, 1999.

Pradubsuwun, D.; Yoneda, T. & Myers, C. (2005) Partial order reduction for detecting safety and timing failures of timed circuits, IEICE Trans. Inf. & Syst., vol. E88-D, no. 7, July 2005.

Toussaint, J.; Simonot-Lion, F. & Thomesse, J.P. (1997). Time constraint verifications methods based on time Petri nets. In Proceedings of the 6th Workshop on Future Trends in Distributed Computing Systems, 1997.

Tripakis, S.; Yovine S. & Bouajjani, A. (2005). Checking Timed Buchi Automata Emptiness Efficiently, Formal Methods in System Design, 26(3), 2005.

Tripakis, S. & Yovine, S. (2001). Analysis of timed systems using time-abstracting bisimulations, Formal Methods in System Design, 18(1), 2001.

Vicario, E. (2001) Static analysis and dynamic steering of time dependent systems, IEEE Transactions on Software Engineering, 2001.

Virbitskaite, I. & Pokozy, E. (1999). A partial order method for the verification of time Petri nets, In Fundamentals of Computation Theory, LNCS 1684, Springer-Verlag, 1999.

Visser, W. & Barringer, H. (2000). Practical CTL model checking - should SPIN be extended? *Software Tools for Technology Transfer, 2(4):350--365, Apr. 2000.*

Yoneda, T. & Ryuba, H. (1998). CTL Model Checking of Time Petri Nets Using Geometric Regions, IEICE Trans. Inf. And Syst., Vol. E99-D, no. 3, 1998.

Yoneda. T & Schlingloff, B.H. (1997). Efficient Verification of Parallel Real-Time Systems, Formal Methods in System Design, Kluwer Academic Publishers, vol. 11, no. 2, pp.187-215, August 1997.

10

A Linear Logic Based Approach to Timed Petri Nets

Norihiro Kamide
Waseda Institute for Advanced Study, 1-6-1 Nishi Waseda, Shinjuku-ku, Tokyo, Japan

1. Introduction

1.1 Relationship between Petri net and linear logic

Petri nets were first introduced by Petri in his seminal Ph.D. thesis, and both the theory and the applications of his model have flourished in concurrency theory (Reisig & Rozenberg, 1998a; Reisig & Rozenberg, 1998b).

The relationships between Petri nets and linear logics have been studied by many researchers (Engberg & Winskel, 1997; Farwer, 1999; Hirai, 2000; Hirai 1999; Ishihara & Hiraish, 2001; Kamide, 2004, Kamide, 2006; Kanovich, 1995; Kanovich 1994; Larchey-Wendling & Galmiche, 1998; Larchey-Wendling & Galmiche, 2000; Lilius, 1992; MartÍ -Oliet & Meseguer, 1991; Okada, 1998; Tanabe, 1997). A category theoretical investigation of such a relationship was given by MartÍ -Oliet and Meseguer (MartÍ -Oliet & Meseguer, 1991), purely syntactical approach using Horn linear logic was established by Kanovich (Kanovich, 1995; Kanovich 1994), a naive phase linear logic for a certain class of Petri nets was given by Okada (Okada, 1998), a linear logical view of object Petri nets were studied by Farwer (Farwer, 1999), and various Petri net interpretations of linear logic using quantale models were obtained by Ishihara and Hiraishi (Ishihara & Hiraish, 2001), Engberg and Winskel (Engberg & Winskel, 1997), Larchey-Wendling and Galmiche (Larchey-Wendling & Galmiche, 1998; Larchey-Wendling & Galmiche, 2000), and Lilius (Lilius, 1992).

Petri net interpretations using Kripke semantics for various fragments and extensions of intuitionistic linear logic were studied by Kamide (Kamide, 2004; Kamide, 2006c). In (Kamide, 2004), Petri net interpretations of various fragments of a spatio-temporal soft linear logic were discussed. In (Kamide, 2006c), Petri nets with inhibitor arcs, which were first introduced by Kosaraju (Kosaraju, 1973) to show the limitation of the usual Petri nets, were described using Kripke semantics for intuitionistic linear logic with strong negation. The approarches using Kripke semantics can obtain a very simple correspondence between Petri net and linear logic.

1.2 Relationship between timed Petri net and temporal linear logic

A number of formalizations of *timed Petri nets* (Bestuzheva and Rudnev, 1994; Wang, 1998) can be considered since time can be associated with tokens, transitions, arcs and places. In the existing linear logic based approaches including the present paper's one, time was associated to tokens (or markings). In fact, to express the fireability of transitions by

multisets of tokens in Petri nets, it seems to be a natural extension to do it by multisets of timed tokens in timed Petri nets.
Temporal linear logic based methods for timed Petri nets were introduced and studied by Tanabe (Tanabe, 1997) and Hirai (Hirai, 1999; Hirai, 2000). In (Tanabe, 1997), a relationship between a timed Petri net and a temporal linear logic was discussed based on quantale models with the soundness theorem for this logic. In (Hirai, 1999; Hirai 2000), a reachability problem for a timed Petri net was solved syntactically by extending Kanovich's result (Kanovich, 1994) with an extended temporal intuitionistic linear logic.
In the present paper, a kind of temporal linear logic, called linear-time linear logic, is used to describe timed Petri nets with timed tokens. This logic is formalized using a natural "linear-time" formalism which is widely used in the standard linear-time temporal logic based on the classical logic rather than linear logics.

1.3 Linear-time temporal logic

Linear-time temporal logic (LTL) has been studied by many researchers, and also been used as a base logic for verifying and specifying concurrent systems (Clarke et al., 1999; Emerson, 1990; Kröger, 1977; Lichtenstein & Pnueli, 2000; Pnueli, 1977; Vardi, 2001; Vardi, 2007) bacause of the virtue of the "linear-time" formalism (Vardi, 2001). LTL is thus known as one of the most useful modal logics based on the classical logic. Sequent calculi for LTL and its neighbors have been introduced by extending the sequent calculus LK for the classical logic (Kawai, 1987; Baratella and Masini, 2004; Paech, 1988; Pliuškevičius, 1991; Szabo, 1980; Szalas, 1986). A sequent calculus LT_ω for LTL was introduced by Kawai, and the cut-elimination and completeness theorems for this calculus were proved (Kawai, 1987). A 2-sequent calculus $2S\omega$ for LTL, which is a natural extension of the usual sequent calculus, was introduced by Baratella and Masini, and the cut-elimination and completeness theorems for this calculus were proved based on an analogy between LTL and Peano arithmetic with ω-rule (Baratella and Masini, 2004). A direct equivalence between Kawai's LT_ω and Baratella and Masini's $2S\omega$ was shown by Kamide introducing the functions that preserve cut-free proofs of these calculi (kamide, 2006b). In the present paper, (intuitionistic) linear logic-based versions of LT_ω and $2S\omega$ are considered.

1.4 Temporal linear logic

Linear logic, which was originally introduced by Girard (Girard, 1987), is known as a resource-aware refinment of the classical and intuitionistic logics, and useful for obtaining more appropriate specifications of concurrent systems (Okada, 1998; Troelstra, 1992). In order to handle both resource-sensitive and time-dependent properties of concurrent systems, combining linear logics with temporal operators has been desired, since the (classical) linear logic (as a basis for temporal logics) is more expressive and appropriate than the classical logic. For this purpose, *temporal linear* logics have been proposed by Hirai (Hirai, 2000), Tanabe (Tanabe, 1997), and Kanovich and Ito (Kanovich & Ito, 1998). Hirai's intuitionistic temporal linear logic (Hirai, 2000) is known as useful for describing a timed Petri net (Hirai, 1999) and a timed linear logic programming language (Tamura et al., 2000). Extensions of Hirai's logic were proposed by Kamide (Kamide, 2004; Kamide, 2006a) as certain *spatio-temporal linear logics* combined with the idea of handling spatiality in Kobayashi, Shimizu and Yonezawa's *modal (spatial) linear logic* (Kobayashi et al., 1999). Tanabe's temporal linear logic (Tanabe, 1997) is used as a base logic for timed Petri net

specifications. Kanovich and Ito's temporal linear logics (Kanovich & Ito, 1998) are a result of combining linear logic with linear-time temporal operators.

1.5 Linear-time linear logic

Linear-time (temporal) linear logics and their usefulness have already been presented by Kanovich and Ito (Kanovich & Ito, 1998). Classical and intuitionistic linear-time linear logics were introduced as cut-free sequent calculi, and the strong completeness theorems for these logics were shown using the algebraic structure of *time phase semantics*. Although in (Kanovich & Ito, 1998), the phase semantic methods for both classical and intuitionistic cases were intensively investigated, other semantic methods and their applications to concurrency theory for the intuitionistic case have yet to be studied sufficiently.
In this paper, an intuitionistic linear-time temporal linear logic, calld also here linear-time linear logic, is introduced as cut-free sequent calculi based on the ideas of Kawai's LT_ω (Kawai, 1987) and Baratella and Masini's $2S\omega$ (Baratella & Masini, 2004). It is shown that the logic based on thses calculi derives intuitive linear-time, informational and Petri net interpretations using Kripke semantics with the completeness theorem. The Kripke semantics presented is introduced based on the exsisting Kripke semantics by Došen (Došen, 1988), Kamide (Kamide, 2003), Kobayashi, Shimizu and Yonezawa (Kobayashi et al., 1999), Hodas and Miller (Hodas & Miller, 1994), Ono and Komori (Ono & Komori, 1985), Urquhart (Urquhart, 1972) and Wansing (Wansing, 1993a; Wansing, 1993b). [1]

1.6 Organization of this paper

This paper is organized as follows.
In Section 2, the linear-time linear logic is introduced as two cut-free Gentzen-type sequent calculi LT and 2LT, and show their equivalence using the method posed in (Kamide, 2006b). The sequent calculi LT and 2LT are regarded as the linear logic based versions of Kawai's LT_ω and Baratella and Masini's $2S\omega$, respectively.
In Section 3, Kripke semantics with a natural timed Petri net interpretation is introduced for LT, and the completeness theorem w.r.t. the semantics is proved as the main result of this paper. The completeness theorem is the basis for obtaining a natural relationship between LT and a timed Petri net.
In Section 4, a timed Petri net with timed tokens is introduced as a structure, and the correspondence between this structure and Kripke frame for LT is observed. An illustrative example for verifying the reachability of timed Petri nets is also addressed based on LT.
In Section 5, this paper is concluded, and some remarkes are given.

2. Linear-time linear logic

2.1 LT

Before the precise discussion, the language used in this paper is introduced. *Formulas* are constructed from propositional variables, **1** (multiplicative constant), $\rightarrow$ (implication), $\wedge$ (conjunction), $*$ (fusion), $!$ (exponential), temporal operators X (next) and G (globally). Lower-case letters *p, q,...* are used for propositional variables, Greek lower-case letters α, β,

[1] For a historical overview of Kripke semantics for modal substructural logics, see. e.g. (Kamide, 2002).

... are used for formulas, and Greek capital letters $\Gamma, \Delta, \ldots$ are used for finite (possibly empty) multisets of formulas. For any $\sharp \in \{!, \mathrm{X}, \mathrm{G}\}$, an expression $\sharp\Gamma$ is used to denote the multiset $\{\sharp\gamma \mid \gamma \in \Gamma\}$. The symbol ≡ is used to denote equality as sequences (or multisets) of symbols. The symbol ω or *N* is used to represent the set of natural numbers. An expression $X^i\alpha$ for any $i \in \omega$ is used to denote $\overbrace{\mathrm{X}\mathrm{X}\cdots\mathrm{X}}^{i}\alpha$, e.g $(\mathrm{X}^0\alpha \equiv \alpha)$, $(\mathrm{X}^1\alpha \equiv \mathrm{X}\alpha)$ and $(\mathrm{X}^{n+1}\alpha \equiv \mathrm{X}^n\mathrm{X}\alpha)$. An expression Γ^* means $\gamma_1 * \cdots * \gamma_n$ $(0 < n)$ if $\Gamma \equiv \{\gamma_1, \cdots, \gamma_n\}$ and means $\emptyset$ if $\Gamma \equiv \emptyset$. An expression $\Delta^\star$ means $\gamma_1 * \cdots * \gamma_n$ if $\Delta \equiv \{\gamma_1, \cdots, \gamma_n\}$ $(0 < n)$ and means 1 if Δ is empty. Lower-case letters i, j and k are used to denote any natural numbers. A sequent is an expression of the form $\Gamma \Rightarrow \gamma$ (the succedent of the sequent is not empty). It is assumed that the terminological conventions regarding sequents (e.g. antecedent, succedent etc.) are the usual ones. If a sequent S is provable in a sequent system L, then such a fact is denoted as $\mathrm{L} \vdash \mathrm{S}$ or $\vdash \mathrm{S}$. The parentheses for $*$ is omitted since $*$ is associative, i.e. $\vdash \alpha * (\beta * \gamma) \Rightarrow (\alpha * \beta) * \gamma$ and $\vdash (\alpha * \beta) * \gamma \Rightarrow \alpha * (\beta * \gamma)$ for any formulas α, β and γ.

In the following, the linear-time linear logic LT is introduced as a sequent calculus. This is regarded as a linear logic version of Kawai's LTω (Kawai, 1987).

Definition 1 (LT) *The initial sequents of* LT *are of the form:*

$$\mathrm{X}^i\alpha \Rightarrow \mathrm{X}^i\alpha \qquad \Rightarrow \mathrm{X}^i\mathbf{1}.$$

The cut rule of LT *is of the form:*

$$\frac{\Gamma \Rightarrow \mathrm{X}^i\alpha \quad \mathrm{X}^i\alpha, \Delta \Rightarrow \gamma}{\Gamma, \Delta \Rightarrow \gamma}\ (\mathrm{cut}).$$

The logical inference rules of LT *are of the form:*

$$\frac{\Gamma \Rightarrow \gamma}{\mathrm{X}^i\mathbf{1}, \Gamma \Rightarrow \gamma}\ (\mathbf{1}\mathrm{we})$$

$$\frac{\Gamma \Rightarrow \mathrm{X}^i\alpha \quad \mathrm{X}^i\beta, \Delta \Rightarrow \gamma}{\mathrm{X}^i(\alpha \to \beta), \Gamma, \Delta \Rightarrow \gamma}\ (\to\mathrm{left}) \qquad \frac{\mathrm{X}^i\alpha, \Gamma \Rightarrow \mathrm{X}^i\beta}{\Gamma \Rightarrow \mathrm{X}^i(\alpha \to \beta)}\ (\to\mathrm{right})$$

$$\frac{\mathrm{X}^i\alpha, \Gamma \Rightarrow \gamma}{\mathrm{X}^i(\alpha \wedge \beta), \Gamma \Rightarrow \gamma}\ (\wedge\mathrm{left1}) \qquad \frac{\mathrm{X}^i\beta, \Gamma \Rightarrow \gamma}{\mathrm{X}^i(\alpha \wedge \beta), \Gamma \Rightarrow \gamma}\ (\wedge\mathrm{left2})$$

$$\frac{\Gamma \Rightarrow \mathrm{X}^i\alpha \quad \Gamma \Rightarrow \mathrm{X}^i\beta}{\Gamma \Rightarrow \mathrm{X}^i(\alpha \wedge \beta)}\ (\wedge\mathrm{right})$$

$$\frac{\mathrm{X}^i\alpha, \mathrm{X}^i\beta, \Gamma \Rightarrow \gamma}{\mathrm{X}^i(\alpha * \beta), \Gamma \Rightarrow \gamma}\ (*\mathrm{left}) \qquad \frac{\Gamma \Rightarrow \mathrm{X}^i\alpha \quad \Delta \Rightarrow \mathrm{X}^i\beta}{\Gamma, \Delta \Rightarrow \mathrm{X}^i(\alpha * \beta)}\ (*\mathrm{right})$$

$$\frac{\mathrm{X}^i\alpha, \Gamma \Rightarrow \gamma}{\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma}\ (!\mathrm{left}) \qquad \frac{\mathrm{X}^i!\Gamma \Rightarrow \mathrm{X}^k\alpha}{\mathrm{X}^i!\Gamma \Rightarrow \mathrm{X}^k!\alpha}\ (!\mathrm{right})$$

$$\frac{\mathrm{X}^i!\alpha, \mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma}{\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma}\ (!\mathrm{co}) \qquad \frac{\Gamma \Rightarrow \gamma}{\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma}\ (!\mathrm{we})$$

$$\frac{\mathrm{X}^{i+k}\alpha, \Gamma \Rightarrow \gamma}{\mathrm{X}^i\mathrm{G}\alpha, \Gamma \Rightarrow \gamma}\ (\mathrm{Gleft}) \qquad \frac{\{\ \Gamma \Rightarrow \mathrm{X}^{i+j}\alpha\ \}_{j\in\omega}}{\Gamma \Rightarrow \mathrm{X}^i\mathrm{G}\alpha}\ (\mathrm{Gright}).$$

It is remarked that (Gright) has infinite premises. It is noted that the cases for $i = k = 0$ in LT derive the usual inference rules for the intuitionistic linear logic.

Although a proof is not given in this paper, the following cut-elimination theorem can be proved by a phase semantic method (Kamide, 2007).

Theorem 2 *(Cut-elimination for* LT*) The rule (cut) is admissible in cut-free* LT.

An expression $\alpha \Leftrightarrow \beta$ means the sequents $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \alpha$. Then, the following sequents are provable in LT for any formulas α, β and any $i \in \omega$:

$$
\begin{array}{l}
X^i\mathbf{1} \Leftrightarrow \mathbf{1}, \\
X^i(\alpha \circ \beta) \Leftrightarrow \mathrm{X}^i\alpha \circ \mathrm{X}^i\beta \quad (\circ \in \{\rightarrow, \wedge, *\}), \\
\mathrm{X}^i!\alpha \Leftrightarrow\ !\mathrm{X}^i\alpha, \\
!\mathrm{G}\alpha \Rightarrow \mathrm{G}!\alpha, \\
\mathrm{G}\alpha \Rightarrow \mathrm{X}\alpha, \\
\mathrm{G}\alpha \Rightarrow \alpha, \\
\mathrm{G}\alpha \Rightarrow \mathrm{GG}\alpha, \\
\mathrm{G}\alpha \Rightarrow \mathrm{XG}\alpha, \\
!\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow !\mathrm{G}\alpha.
\end{array}
$$

The last sequent above corresponds to the linear logic version of the temporal induction axiom: $\alpha \rightarrow (\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \rightarrow \mathrm{G}\alpha)$, and an LT-proof of this sequent is as follows. .

$$
\dfrac{\dfrac{\dfrac{\vdots}{\{\ \alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^k\alpha\ \}_{k\in\omega}}\ (\mathrm{Gright})}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{G}\alpha}\ (!\mathrm{left})}{!\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{G}\alpha}\ (!\mathrm{right})
$$

$$
\dfrac{!\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow !\mathrm{G}\alpha}{}
$$

where $\vdash \alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^k\alpha$ for any $k \in \omega$ is shown by mathematical induction on k as follows. The base step, i.e. $k = 0$, is obvious using (!we). The induction step can be shown using (!co) as follows.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \text{ind.hyp.}}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^k\alpha \quad \mathrm{X}^{k+1}\alpha \Rightarrow \mathrm{X}^{k+1}\alpha}\ (\rightarrow\mathrm{left})}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha), \mathrm{X}^k(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^{k+1}\alpha}\ (\mathrm{Gleft})}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha), \mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^{k+1}\alpha}\ (!\mathrm{left})}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha), !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^{k+1}\alpha}\ (!\mathrm{co}).
$$

$$
\dfrac{}{\alpha, !\mathrm{G}(\alpha \rightarrow \mathrm{X}\alpha) \Rightarrow \mathrm{X}^{k+1}\alpha}
$$

2.2 2LT

A 2-sequent calculus 2LT for the linear-time linear logic is introduced below. This calculus is a linear logic version of Baratella and Masini's 2-sequent calculus 2Sω (Baratella & Masini, 2004). The language of 2LT and the notations used are almost the same as those of LT.

Definition 3 *An expression α^i (a is a formula and $i \in \omega$) is called an indexed formula. Let γ be an indexed formula and Γ be finite (possibly empty) multiset of indexed formulas. Then an expression $\Gamma \Rightarrow^2 \gamma$ is called a 2-sequent.*

An expression Γ^i is used to denote the multiset of i-indexed formulas.

Definition 4 (2LT) *The initial sequents of* 2LT *are of the form:*

$$\alpha^i \Rightarrow^2 \alpha^i \qquad \Rightarrow^2 \mathbf{1}^i.$$

The cut rule of 2LT is of the form:

$$\frac{\Gamma \Rightarrow^2 \alpha^i \quad \alpha^i, \Delta \Rightarrow^2 \gamma}{\Gamma, \Delta \Rightarrow^2 \gamma} \ (\text{cut2}).$$

The logical inference rules of 2LT are of the form:

$$\frac{\Gamma \Rightarrow^2 \gamma}{\mathbf{1}^i, \Gamma \Rightarrow^2 \gamma} \ (\mathbf{1}\text{we2})$$

$$\frac{\Gamma \Rightarrow^2 \alpha^i \quad \beta^i, \Delta \Rightarrow^2 \gamma}{(\alpha \rightarrow \beta)^i, \Gamma, \Delta \Rightarrow^2 \gamma} \ (\rightarrow\text{left2}) \qquad \frac{\alpha^i, \Gamma \Rightarrow^2 \beta^i}{\Gamma \Rightarrow^2 (\alpha \rightarrow \beta)^i} \ (\rightarrow\text{right2})$$

$$\frac{\alpha^i, \Gamma \Rightarrow^2 \gamma}{(\alpha \wedge \beta)^i, \Gamma \Rightarrow^2 \gamma} \ (\wedge\text{left12}) \qquad \frac{\beta^i, \Gamma \Rightarrow^2 \gamma}{(\alpha \wedge \beta)^i, \Gamma \Rightarrow^2 \gamma} \ (\wedge\text{left22})$$

$$\frac{\Gamma \Rightarrow^2 \alpha^i \quad \Gamma \Rightarrow^2 \beta^i}{\Gamma \Rightarrow^2 (\alpha \wedge \beta)^i} \ (\wedge\text{right2})$$

$$\frac{\alpha^i, \beta^i, \Gamma \Rightarrow^2 \gamma}{(\alpha * \beta)^i, \Gamma \Rightarrow^2 \gamma} \ (*\text{left2}) \qquad \frac{\Gamma \Rightarrow^2 \alpha^i \quad \Delta \Rightarrow^2 \beta^i}{\Gamma, \Delta \Rightarrow^2 (\alpha * \beta)^i} \ (*\text{right2})$$

$$\frac{\alpha^i, \Gamma \Rightarrow^2 \gamma}{(!\alpha)^i, \Gamma \Rightarrow^2 \gamma} \ (!\text{left2}) \qquad \frac{(!\Gamma)^i \Rightarrow^2 \alpha^k}{(!\Gamma)^i \Rightarrow^2 (!\alpha)^k} \ (!\text{right2})$$

$$\frac{(!\alpha)^i, (!\alpha)^i, \Gamma \Rightarrow^2 \gamma}{(!\alpha)^i, \Gamma \Rightarrow^2 \gamma} \ (!\text{co2}) \qquad \frac{\Gamma \Rightarrow^2 \gamma}{(!\alpha)^i, \Gamma \Rightarrow^2 \gamma} \ (!\text{we2})$$

$$\frac{\alpha^{i+k}, \Gamma \Rightarrow^2 \gamma}{(\mathrm{G}\alpha)^i, \Gamma \Rightarrow^2 \gamma} \ (\text{Gleft2}) \qquad \frac{\{\ \Gamma \Rightarrow \alpha^{i+j}\ \}_{j \in \omega}}{\Gamma \Rightarrow^2 (\mathrm{G}\alpha)^i} \ (\text{Gright2})$$

$$\frac{\alpha^{i+1}, \Gamma \Rightarrow^2 \gamma}{(\mathrm{X}\alpha)^i, \Gamma \Rightarrow^2 \gamma} \ (\text{Xleft}) \qquad \frac{\Gamma \Rightarrow^2 \alpha^{i+1}}{\Gamma \Rightarrow^2 (\mathrm{X}\alpha)^i} \ (\text{Xright}).$$

An expression $L \vdash \Gamma \Rightarrow^2 \gamma$ is used to denote the fact that $\Gamma \Rightarrow^2 \gamma$ is provable in a 2-sequent calculus *L*.

Definition 5 *Let* $\mathcal{L}_1$ *be the set of formulas of* LT *and* $\mathcal{L}_2$ *be the set of indexed formulas of 2LT.*

A function f from $\mathcal{L}_1$ *to* $\mathcal{L}_2$ *is defined by* $f(\mathrm{X}^i\alpha) := \alpha^i$ *for any formula* α.
A function g from $\mathcal{L}_2$ *to* $\mathcal{L}_1$ *is defined by* $g(\alpha^i) := X^i\alpha$ *for any formula* α.

It is remark that $fg(\alpha^i) = \alpha^i$ and $gf(X^i\alpha) = X^i\alpha$ hold for any formula α.

Theorem 6 (Equivalence between LT and 2LT) (1) *for any 2-sequent* $\Gamma \Rightarrow^2 \gamma$, *if* 2LT $\vdash \Gamma \Rightarrow^2 \gamma$, then LT $\vdash g(\Gamma) \Rightarrow g(\gamma)$. (2) *for any sequent* $\Gamma \Rightarrow \gamma$, *if* LT $-$ (cut) $\vdash \Gamma \Rightarrow \gamma$, *then* 2LT $-$ (cut2) $\vdash \Gamma \Rightarrow^2 \gamma$.

Proof We show only (1) by induction on a proof *P* of $\Gamma \Rightarrow^2 \gamma$ in 2LT. We show only the following case.

Case (Xleft): The last inference of P is of the form:

$$\frac{\alpha^{i+1}, \Sigma \Rightarrow^2 \gamma}{(\mathrm{X}\alpha)^i, \Sigma \Rightarrow^2 \gamma}\ (\mathrm{Xleft}).$$

By the hypothesis of induction, we obtain LT $\vdash g(\alpha^{i+1}), g(\Sigma) \Rightarrow g(\gamma)$, and hence obtain LT $\vdash g((\mathrm{X}\alpha)^i), g(\Sigma) \Rightarrow g(\gamma)$ by $g(\alpha^{i+1}) = \mathrm{X}^{i+1}\alpha = X^i(\mathrm{X}\alpha) = g((\mathrm{X}\alpha)^i)$. Q.E.D.

By Theorems 2 and 6, the following theorem is obtained.

Theorem 7 (Cut-elimination for 2LT) *The rule* (cut2) *is admissible in cut-free* 2LT.

Proof Suppose 2LT $\vdash \Gamma \Rightarrow^2 \gamma$ for a 2-sequent $\Gamma \Rightarrow^2 \gamma$. Then we have LT $\vdash g(\Gamma) \Rightarrow g(\gamma)$ by Theorem 6 (1). By Theorem 2, we obtain $\mathrm{LT} - (\mathrm{cut}) \vdash g(\Gamma) \Rightarrow g(\gamma)$. We thus obtain $2\mathrm{LT} - (\mathrm{cut2}) \vdash fg(\Gamma) \Rightarrow^2 fg(\gamma)$ by Theorem 6 (2). Therefore $2\mathrm{LT} - (\mathrm{cut2}) \vdash \Gamma \Rightarrow^2 \gamma$.
Conversely, by Theorem 7 and an appropriate modification of Theorem 6, a proof of Theorem 2 is also derived. Q.E.D.

3. Kripke semantics

3.1 Kripke model and soundness

The following definition (except the existence of N) of the Kripke frame is the same as that for the (fragment of) intuitionistic linear logic (Kamide, 2003).

Definition 8 A Kripke frame *for* LT *is a structure* $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *satisfying the following conditions:*

1. N *is the set of natural numbers,*
2. $\langle M, \cdot, \varepsilon \rangle$ *is a commutative monoid with the identity* ε*,*
3. $\langle M, \geq \rangle$ *is a pre-ordered set,*
4. $\dagger$ *is a unary operation on* M *such that*

$$\begin{array}{ll} \mathrm{C0}: & \varepsilon \geq \dagger\varepsilon, \\ \mathrm{C1}: & \dagger x \geq x \text{ for all } x \in M, \\ \mathrm{C2}: & \dagger x \geq \dagger\dagger x \text{ for all } x \in M, \\ \mathrm{C3}: & \dagger x \cdot \dagger y \geq \dagger(x \cdot y) \text{ for all } x, y \in M, \\ \mathrm{C4}: & \dagger x \geq \dagger x \cdot \dagger x \text{ for all } x \in M, \\ \mathrm{C5}: & \dagger y \cdot x \geq x \text{ for all } x, y \in M, \end{array}$$

5. $\cdot$ *is monotonic with respect to* $\geq$*, i.e.*

$$\mathrm{C6}: \quad y \geq z \text{ implies } x \cdot y \geq x \cdot z \text{ for all } x, y, z \in M.$$

Definition 9 *A valuation* $\models$ *on a Kripke frame* $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *for* LT is *a mapping from the set of all propositional variables to the power set of* $M \times N$ *and satisfying the following hereditary condition:* $(x, i) \in \models (p)$ *and* $y \geq x$ *imply* $(y, i) \in \models (p)$*for any propositional variable* p, any $i \in N$ *and any* $x, y \in M$. *An expression* $(x, i) \models p$ *will be used for* $(x, i) \in \models (p)$. *Each valuation* $\models$ *can be extended to a mapping from the set of all formulas to the power set of* $M \times N$ *by*

1. $(x, i) \models \mathbf{1}$ *iff* $x \geq \varepsilon$,
2. $(x, i) \models \alpha \rightarrow \beta$ *iff* $(y, i) \models \alpha$ *implies* $(x \cdot y, i) \models \beta$ *for all* $y \in M$,
3. $(x, i) \models \alpha \wedge \beta$ *iff* $(x, i) \models \alpha$ *and* $(x, i) \models \beta$,
4. $(x, i) \models \alpha * \beta$ *iff* $(y, i) \models \alpha$ *and* $(z, i) \models \beta$ *for some* $y, z \in M$ *with* $x \geq y \cdot z$,
5. $(x, i) \models !\alpha$ *iff* $y \models \alpha$ *for some* $(y, i) \in M$ *with* $x \geq \dagger y$,

6. $(x,i) \models \mathrm{X}\alpha$ *iff* $(x,i+1) \models \alpha$,
7. $(x,i) \models \mathrm{G}\alpha$ *iff* $(x,l) \models \alpha$ *for all* $l \in N$ *with* $l \geq i$.

Proposition 10 *Let* $\models$ *be a valuation on a Kripke frame* $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *for* LT. *Then the following hereditary condition holds:* $(x,i) \models \alpha$ *and* $y \geq x$ *imply* $(y,i) \models \alpha$ *for any formula* α, *any* $i \in N$, *and any* $x, y \in M$.

Proof By induction on the complexity of α. Q.E.D.

Definition 11 *A Kripke model for LT is a structure* $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ *such that*

1. $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *is a Kripke frame for* LT,
2. $\models$ is a valuation on $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$.

A formula α *is true in a Kripke model* $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ *for* LT *if* $(\varepsilon, 0) \models \alpha$, and valid in a *Kripke frame* $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *for* LT *if it is true for any valuation* $\models$ *on the Kripke frame. A sequent* $\alpha_1, \cdots, \alpha_n \Rightarrow \beta$ *(or* $\Rightarrow \beta$*) is* true *in a Kripke model* $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ *for* LT *if the formula* $\alpha_1 * \cdots * \alpha_n \rightarrow \beta$ *(or* β *respectively) is true in it, and valid in a Kripke frame for* LT *if the formula* $\alpha_1 * \cdots * \alpha_n \rightarrow \beta$ *(or* β *respectively)is valid in it.*

The Kripke model $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ defined has a natural informational interpretation due to Urquhart (Urquhart, 1972) and Wansing (Wansing, 1993a; Wansing, 1993b). *M* is a set of information pieces, $\cdot$ is the addition of information pieces, $\dagger$ is the infinite addition of information pieces, and ε is the empty piece of information. Then the forcing relation $(x,i) \models \alpha$ can read as "the resource α is obtained at the time *i* by using the information piece *x*."

Theorem 12 (Soundness) *Let* C *be a class of* Kripke *frames for* LT, $L := \{S \mid \mathrm{LT} \vdash S\}$ and $L(C) := \{S \mid S \text{ is valid in all frames of } C\}$. *Then* $L \subseteq L(C)$

Proof It is sufficient to prove the following: for any sequent S, if S is provable, then S is valid in any frame $F := \langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle \in C$. This is proved by induction on a proof *P* of *S*. We distinguish the cases according to the last inference rules and initial sequents in *P* . Let $\models$ be a valuation on *F* . In the following, we sometimes use implicitly the fact that $\geq$ is a pre-order, $\langle M, \cdot, \varepsilon \rangle$ is a commutative monoid with the identity ε, $\cdot$ is monotonic, and $\models$ has the hereditary condition (Proposition 10). We show some cases.

Case (!left): It is shown that *L(C)* is closed under (!left), i.e. for any formula α and any multiset Γ of formulas, if $\mathrm{X}^i\alpha, \Gamma \Rightarrow \gamma$is valid in F then so is $\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma$. In the following, we consider only the case that Γ is nonempty (the empty case can be shown similarly). Suppose that $(1)\ (\varepsilon, 0) \models \mathrm{X}^i\alpha * \Gamma^* \rightarrow \gamma$ and $(2)\ (x,0) \models \mathrm{X}^i!\alpha * \Gamma^*$ for any $x \in M$. We will show $(x,0) \models \gamma$. By (2), there exist $x_1, x_2 \in M$ such that $(3)\ x \geq x_1 \cdot x_2$, $(4)\ (x_1, i) \models !\alpha$ and $(5)\ (x_2, 0) \models \Gamma^*$. By (4), there exists $x_1' \in M$ such that $(6)\ x_1 \geq \dagger x_1'$ and $(7)\ (x_1', i) \models \alpha$. By (6), the frame condition C1 and the transitivity of $\geq$, we have $(8)\ x_1 \geq x_1'$. Moreover, by (8) and the monotonicity of $\cdot$, we have $(9)\ x_1 \cdot x_2 \geq x_1' \cdot x_2$. By (9), (3) and the transitivity of $\geq$, we have $(10)\ x \geq x_1' \cdot x_2$. Thus, by (10), (7) and (5), we obtain the following: there exist $x_1', x_2 \in M$ such that $x \geq x_1' \cdot x_2$, $(x_1', 0) \models \mathrm{X}^i\alpha$ and $(x_2, 0) \models \Gamma^*$. Hence, by (1) we have $(x,0) \models \gamma$.

Case (!right): It is shown that *L(C)* is closed under (!right), i.e. for any formula α and any multiset Γ of formulas, if $\mathrm{X}^i!\Gamma \Rightarrow \mathrm{X}^i\alpha$ is valid in *F* then so is $\mathrm{X}^i!\Gamma \Rightarrow \mathrm{X}^i!\alpha$.

We only show the case that Γ is nonempty (the empty case can easily be shown using the frame condition C0). Suppose (1) $(x,0) \models (\mathrm{X}^i!\Gamma)^*$ $(\Gamma \equiv \{\gamma_1, \cdots, \gamma_n\}\ (0 < n))$ for any

$x \in M$ and (2) $(\varepsilon, 0) \models (\mathrm{X}^i!\Gamma)^* \rightarrow \mathrm{X}^i\alpha$. We will show $(x, 0) \models \mathrm{X}^i!\alpha$. By (1), we have that there exist $x_1, \cdots, x_n \in M$ such that (3) $x \geq x_1 \cdots x_n$, and (4) $(x_1, i) \models !\gamma_1, \cdots, (x_n, i) \models !\gamma_n$. Then, by (4), we have that for any $j \in \{1, \cdots, n\}$, there exists $x'_j \in M$ such that (5) $x_j \geq \dagger x'_j$ and (6) $(x'_j, i) \models \gamma_j$. By (6), the frame condition C1 and the hereditary condition of $\models$, we obtain (7) $(\dagger x'_j, i) \models \gamma_j$. Thus we have that there exists $\dagger x'_j \in M$ (because M is closed under $\dagger$, and there exists $x'_j \in M$) such that $\dagger x'_j \geq \dagger\dagger x'_j$ (by the frame condition C2) and $(\dagger x'_j, i) \models \gamma_j$ (by (7)). This means that (8) $(\dagger x'_j, i) \models !\gamma_i$, i.e. $(\dagger x'_j, 0) \models X^i!\gamma_i$ for any $j \in \{1, \cdots, n\}$. Further we have (9) $\dagger x'_1 \cdots \dagger x'_n \geq \dagger x'_1 \cdots \dagger x'_n$ since $\geq$ is reflexive. Hence we have that there exist $\dagger x_1, \cdots, \dagger x_n \in M$ such that (8) and (9). This means $(\dagger x'_1 \cdots \dagger x'_n, 0) \models \mathrm{X}^i!\gamma_1 * \cdots * \mathrm{X}^i!\gamma_n$, i.e. (10) $(\dagger x'_1 \cdots \dagger x'_n, 0) \models (\mathrm{X}^i!\Gamma)^*$. By the hypothesis (2) and the fact (10), we have (11) $(\dagger x'_1 \cdots \dagger x'_n, 0) \models \mathrm{X}^i\alpha$, i.e. $(\dagger x'_1 \cdots \dagger x'_n, i) \models \alpha$. By the facts (3), (5), the monotonicity of $\cdot$ and the frame conditions C2, C3, we have (12) $x \geq x_1 \cdots x_n \geq \dagger x'_1 \cdots \dagger x'_n \geq \dagger\dagger x'_1 \cdots \dagger\dagger x'_n \geq \dagger(\dagger x'_1 \cdots \dagger x'_n)$.

Hence we obtain the following: there exist $\dagger x'_1 \cdots \dagger x'_n \in M$ (because M is closed under $\cdot$) such that $x \geq \dagger(\dagger x'_1 \cdots \dagger x'_n)$ (by (12) and the transitivity of $\geq$) and $(\dagger x'_1 \cdots \dagger x'_n, i) \models \alpha$ (by (11)). This means $(x, i) \models !\alpha$, i.e. $(x, 0) \models \mathrm{X}^i!\alpha$.

Case (!co)): It is shown that *L(C)* is closed under (!co), i.e. for any formulas α, γ and any multiset Γ of formulas, if $\mathrm{X}^i!\alpha, \mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma$ is valid in *F* then so is $\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma$. In the following we consider only the case that Γ is nonempty (the empty case can be shown similarly). Suppose (1) $(x, 0) \models \mathrm{X}^i!\alpha * \Gamma^*$ for any $x \in M$ and (2) $(\varepsilon, 0) \models \mathrm{X}^i!\alpha * \mathrm{X}^i!\alpha * \Gamma^* \rightarrow \gamma$. We will show $(x, 0) \models \gamma$. By (1), there exist $x_1, x_2 \in M$ such that (3) $x \geq x_1 \cdot x_2$, (4) $(x_1, i) \models !\alpha$ and (5) $(x_2, 0) \models \Gamma^*$. By (4), we have that there exists $x'_1 \in M$ such that (6) $x_1 \geq \dagger x'_1$ and (7) $(x'_1, i) \models \alpha$. By (3), (6) and the monotonicity of $\cdot$, we have (8) $x \geq x_1 \cdot x_2 \geq \dagger x'_1 \cdot x_2$. On the other hand, we have that there exists $\dagger x'_1 \in M$ such that $\dagger x'_1 \geq \dagger x'_1 \cdot \dagger x'_1$ (by the frame condition C4), $(\dagger x'_1, i) \models !\alpha$ (because, by (7), the frame conditions C1, C2 the hereditary condition of $\models$, we have that there exists $\dagger x'_1 \in M$ such that $\dagger x'_1 \geq \dagger\dagger x'_1$ and $(\dagger x'_1, i) \models \alpha$). This means (9) $(\dagger x'_1, 0) \models \mathrm{X}^i!\alpha * \mathrm{X}^i!\alpha$. Further we have that there exist $\dagger x'_1, x_2 \in M$ such that $x \geq \dagger x'_1 \cdot x_2$ (by (8) and the transitivity of $\geq$), $(\dagger x'_1, 0) \models \mathrm{X}^i!\alpha * \mathrm{X}^i!\alpha$ (by (9)) and $(x_2, 0) \models \mathrm{X}^i\Gamma^*$ (by (5)). This means (10) $(x, 0) \models \mathrm{X}^i!\alpha * \mathrm{X}^i!\alpha * \Gamma^*$. By the hypothesis (2) and the fact (10), we obtain $(x, 0) \models \gamma$.

Case (!we): It is shown that *L(C)* is closed under (!we), i.e. for any formulas α, γ and any multiset Γ of formulas, if $\Gamma \Rightarrow \gamma$ is valid in *F* then so is $\mathrm{X}^i!\alpha, \Gamma \Rightarrow \gamma$. In the following we consider only the case that Γ is nonempty (the empty case can be shown similarly). Suppose (1) $(x, 0) \models \mathrm{X}^i!\alpha * \Gamma^*$ for any $x \in M$ and (2) $(\varepsilon, 0) \models \Gamma^* \rightarrow \gamma$. We will show $(x, 0) \models \gamma$. By (1), we have that there exist $x_1, x_2 \in M$ such that (3) $x \geq x_1 \cdot x_2$, (4) $(x_1, i) \models !\alpha$ and (5) $(x_2, 0) \models \Gamma^*$. By (4), we have that there exists $x'_1 \in M$ such that (6) $x_1 \geq \dagger x'_1$ and $(x'_1, i) \models \alpha$. Then we obtain (7) $x \geq x_2$ since we have $x \geq x_1 \cdot x_2 \geq \dagger x'_1 \cdot x_2 \geq x_2$ by (3), (6), the monotonicity of $\cdot$, the transitivity of $\geq$ and the frame condition C5. Hence, by (7), (5) and the hereditary condition of $\models$, we obtain (8) $(x, 0) \models \Gamma^*$. Thus we obtain $(x, 0) \models \gamma$ by the hypothesis (2) and the fact (8).

Case (Gleft): It is shown that *L(C)* is closed under (Gleft), i.e. for any formulas α and multiset Γ of formulas, if $\mathrm{X}^{i+k}\alpha, \Gamma \Rightarrow \gamma$ is valid in *F* thenso is $\mathrm{X}^i\mathrm{G}\alpha, \Gamma \Rightarrow \gamma$. In the following, we consider only the case that Γ is nonempty (the empty case can be shown

similarly).
Suppose $(\varepsilon,0) \models \mathrm{X}^{i+k}\alpha * \Gamma^* \rightarrow \gamma$, i.e., $\forall y \in M$ [$\exists y_1, y_2 \in M$($y \geq y_1 \cdot y_2$ and $(y_1,0) \models \mathrm{X}^{i+k}\alpha$ and $(y_2,0) \models \Gamma^*$) implies $(y,0) \models \gamma$]. We will show $(\varepsilon,0) \models \mathrm{X}^i\mathrm{G}\alpha * \Gamma^* \rightarrow \gamma$, i.e., $\forall y \in M$ [$\exists y_1, y_2 \in M$ ($y \geq y_1 \cdot y_2$ and $(y_1,0) \models \mathrm{X}^i\mathrm{G}\alpha$ and $(y_2,0) \models \Gamma^*$) implies $(y,0) \models \gamma$]. It is thus enough to show that $(y_1,0) \models \mathrm{X}^i\mathrm{G}\alpha$ implies $(y_1,0) \models \mathrm{X}^{i+k}\alpha$. Suppose $(y_1,0) \models \mathrm{X}^i\mathrm{G}\alpha$. Then $(y_1,0) \models \mathrm{X}^i\mathrm{G}\alpha$ iff $(y_1,i) \models \mathrm{G}\alpha$ iff $\forall j \geq i$ $[(y_1,j) \models \alpha]$ iff $\forall l \in N$ $[(y_1,i+l) \models \alpha]$. Thus we obtain $(y_1,0) \models \mathrm{X}^{i+k}\alpha$.
Case (Gright): It is shown that *L(C)* is closed under (Gright), i.e. for any formula α and multiset Γ of formulas, if $\Gamma \Rightarrow \mathrm{X}^{i+j}\alpha$ (for all $j \in \omega$) are valid in *F* then so is $\Gamma \Rightarrow \mathrm{X}^i\mathrm{G}\alpha$. We consider here only the case that Γ is nonempty (the empty case can be shown similarly). Suppose $(\varepsilon,0) \models \Gamma^* \rightarrow \mathrm{X}^{i+j}\alpha$ for all $j \in \omega$, i.e. for all $j \in \omega$, $\forall y \in M$ [$(\varepsilon,0) \models \Gamma^*$ implies $(y,0) \models \mathrm{X}^{i+j}\alpha$]. We will show $(\varepsilon,0) \models \Gamma^* \rightarrow \mathrm{X}^i\mathrm{G}\alpha$, i.e., $\forall y \in M$ [$(\varepsilon,0) \models \Gamma^*$ implies $(y,0) \models \mathrm{X}^i\mathrm{G}\alpha$]. It is thus enough to show that $(y,0) \models \mathrm{X}^{i+j}\alpha$ for all $j \in \omega$ implies $(y,0) \models \mathrm{X}^i\mathrm{G}\alpha$. Suppose $(y,0) \models \mathrm{X}^{i+j}\alpha$ for all $j \in \omega$, i.e. $(y,i+j) \models \alpha$ for all $j \in \omega$. Then we obtain $(y,0) \models \mathrm{X}^i\mathrm{G}\alpha$ as follows: $\forall j \in \omega$ [$(y,i+j) \models \alpha$] iff $\forall k \geq i$ [$(y,k) \models \alpha$] iff $(y,i) \models \mathrm{G}\alpha$ iff $(y,0) \models \mathrm{X}^i\mathrm{G}\alpha$. Q.E.D.

3.2 Completeness

In order to prove the completeness theorem, constructing a canonical model is needed, and the resulting canonical model will be used to show the relationship between a timed Petri net and LT.
Definition 13 A canonical model is a structure $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ such that

1. *N is the set of natural numbers,*
2. $M := \{\Gamma \mid \Gamma$ *is a finite multiset of formulas*$\}$,
3. *for any* $\Gamma, \Delta \in M$, $\Gamma \cdot \Delta := \Gamma \cup \Delta$ *(the multiset union), where* $\Gamma \cdot \Delta$ *will also be denoted as* Γ, Δ,
4. *for any* $\Gamma \in M$, $\dagger\Gamma := !\Gamma =$ *the multiset* $\{!\gamma \mid \gamma \in \Gamma\}$,
5. ε *is the empty multiset,*
6. *for any* $\Gamma, \Delta \in M$, $\Gamma \geq \Delta$ *is defined by* $\vdash \Gamma \Rightarrow \Delta^\star$ *where* $\Delta^\star \equiv \gamma_1 * \cdots * \gamma_n$ *if* $\Delta \equiv \{\gamma_1, \cdots, \gamma_n\}$ $(0 < n)$, *and* $\Delta^\star \equiv \mathbf{1}$ *if* Δ *is empty,*
7. *a valuation* $\models$ *on* $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ *is a mapping from the set* PROP *of all propositional variables to the power set of* $M \times N$ *defined by*

   $(\Gamma, i) \in \models (p)$ *iff* $\vdash \Gamma \Rightarrow \mathrm{X}^i p$, *for any* $p \in$ PROP, *any* $\Gamma \in M$ *and any* $i \in N$.

It can then be shown that $\langle M, N, \cdot, \dagger, \varepsilon, \geq \rangle$ is a Kripke frame for LT. It is remarked that the condition C0 corresponds to $\vdash \Rightarrow \mathbf{1}$ since $\varepsilon \geq \dagger\varepsilon$ or $\varepsilon \geq \varepsilon$ is defined by $\{\} \Rightarrow \{\}^\star$ where {} is the empty multiset. It is also remarked that the sequent $\Rightarrow$ is not true in this canonical model. $\{\} \geq \{\}$ is interpreted as $\vdash \Rightarrow \mathbf{1}$ (i.e., $\{\} \Rightarrow \{\}^\star$) but $\{\} \Rightarrow \{\}$ does not correspond to $\Rightarrow \mathbf{1}$. Also $\Rightarrow$ is not true in any Kripke model, because $\Rightarrow$ is interpreted as $\{\} \Rightarrow \{\}^{\star\star}$ (i.e., $\{\} \Rightarrow \{\}$).
Further it will be proved that $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ is a Kripke model for LT. To show this fact is essentially to show the completeness theorem. To achive the completeness theorem, the following lemma is needed.

Lemma 14 *Let* $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ *be the canonical model defined in Definition 13. Then, for any formula* γ, *any* $\Gamma \in M$ *and any* $i \in N$,

$$(\Gamma, i) \models \gamma \text{ if and only if } \mathrm{LT} \vdash \Gamma \Rightarrow \mathrm{X}^i\gamma.$$

Proof This lemma is proved by induction on the complexity of γ. We show some cases.
(Case $\gamma \equiv p$) : By the definition of $\models$.
(Case $\gamma \equiv \mathbf{1}$): Suppose $(\Gamma, i) \models \mathbf{1}$. Then we have $(\Gamma, i) \models \mathbf{1}$ iff $\Gamma \geq \{\}$ iff $\vdash \Gamma \Rightarrow \{\}^*$iff $\vdash \Gamma \Rightarrow \mathbf{1}$. Thus we obtain $\vdash \Gamma \Rightarrow \mathrm{X}^i\mathbf{1}$:

$$\dfrac{\Gamma \Rightarrow \mathbf{1} \quad \dfrac{}{\mathbf{1} \Rightarrow \mathrm{X}^i\mathbf{1}}\,(\mathbf{1}\text{we})}{\Gamma \Rightarrow \mathrm{X}^i\mathbf{1}}\,(\text{cut}).$$

Conversely, suppose $\vdash \Gamma \Rightarrow \mathrm{X}^i\mathbf{1}$. Then we have

$$\dfrac{\Gamma \Rightarrow \mathrm{X}^i\mathbf{1} \quad \dfrac{}{\mathrm{X}^i\mathbf{1} \Rightarrow \mathbf{1}}\,(\mathbf{1}\text{we})}{\Gamma \Rightarrow \mathbf{1}}\,(\text{cut}),$$

and hence $(\Gamma, i) \models \mathbf{1}$
(Case $\gamma \equiv \alpha_1 \rightarrow \alpha_2$): First we show that $(\Gamma, i) \models \alpha_1 \rightarrow \alpha_2$ implies $\vdash \Gamma \Rightarrow \mathrm{X}^i\,\alpha_1 \rightarrow \alpha_2$. Suppose $(\Gamma, i) \models \alpha_1 \rightarrow \alpha_2$ i.e. $(\Delta, i) \models \alpha_1$ implies $(\Gamma \cup \Delta, i) \models \alpha_2$ for any $\Delta \in M$. We take $\{\mathrm{X}^i\alpha_1\}$for Δ. We have $(\{\mathrm{X}^i\alpha_1\}, i) \models \alpha_1$ by the induction hypothesis, and $(\Gamma \cup \{\mathrm{X}^i\alpha_1\}, i) \models \alpha_2$ by the hypothesis. Thus we have $\vdash \Gamma, \mathrm{X}^i\alpha_1 \Rightarrow \mathrm{X}^i\alpha_2$ by the induction hypothesis, and hence $\vdash \Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \rightarrow \alpha_2)$ by $(\rightarrow\text{right})$. Conversely, suppose $\vdash \Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \rightarrow \alpha_2)$ and $(\Delta, i) \models \alpha_1$ for any $\Delta \in M$. Then we have $\vdash \Delta \Rightarrow \mathrm{X}^i\alpha_1$ by the induction hypothesis. We obtain:

$$\dfrac{\Delta \Rightarrow \mathrm{X}^i\alpha_1 \quad \dfrac{\Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \rightarrow \alpha_2) \quad \dfrac{\mathrm{X}^i\alpha_1 \Rightarrow \mathrm{X}^i\alpha_1 \quad \mathrm{X}^i\alpha_2 \Rightarrow \mathrm{X}^i\alpha_2}{\mathrm{X}^i(\alpha_1 \rightarrow \alpha_2), \mathrm{X}^i\alpha_1 \Rightarrow \mathrm{X}^i\alpha_2}\,(\rightarrow\text{left})}{\Gamma, \mathrm{X}^i\alpha_1 \Rightarrow \mathrm{X}^i\alpha_2}\,(\text{cut})}{\Gamma, \Delta \Rightarrow \mathrm{X}^i\alpha_2}\,(\text{cut})$$

and hence $(\Gamma \cup \Delta, i) \models \alpha_2$ by the induction hypothesis.
(Case $\gamma \equiv \alpha_1 \wedge \alpha_2$): First we show that $(\Gamma, i) \models \alpha_1 \wedge \alpha_2$ implies $\vdash \Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \wedge \alpha_2)$ for any $\Gamma \in M$. Suppose $(\Gamma, i) \models \alpha_1 \wedge \alpha_2$. Then we have $(\Gamma, i) \models \alpha_1$ and $(\Gamma, i) \models \alpha_2$. We obtain $\vdash \Gamma \Rightarrow \mathrm{X}^i\alpha_1$ and $\vdash \Gamma \Rightarrow \mathrm{X}^i\alpha_2$ by the hypothesis of induction. Thus we have $\vdash \Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \wedge \alpha_2)$ by $(\wedge\text{right})$. Conversely, suppose $\vdash \Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \wedge \alpha_2)$. Then we have

$$\dfrac{\Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \wedge \alpha_2) \quad \dfrac{\mathrm{X}^i\alpha_1 \Rightarrow \mathrm{X}^i\alpha_1}{\mathrm{X}^i(\alpha_1 \wedge \alpha_2) \Rightarrow \mathrm{X}^i\alpha_1}\,(\wedge\text{left1})}{\Gamma \Rightarrow \mathrm{X}^i\alpha_1}\,(\text{cut}),$$

$$\dfrac{\Gamma \Rightarrow \mathrm{X}^i(\alpha_1 \wedge \alpha_2) \quad \dfrac{\mathrm{X}^i\alpha_2 \Rightarrow \mathrm{X}^i\alpha_2}{\mathrm{X}^i(\alpha_1 \wedge \alpha_2) \Rightarrow \mathrm{X}^i\alpha_2}\,(\wedge\text{left2})}{\Gamma \Rightarrow \mathrm{X}^i\alpha_2}\,(\text{cut}).$$

Thus we have $(\Gamma, i) \models \alpha_1$ and $(\Gamma, i) \models \alpha_2$ by the induction hypothesis, and hence $(\Gamma, i) \models \alpha_1 \wedge \alpha_2$.

(Case $\gamma \equiv !\beta$): First, we show that $(\Gamma, i) \models !\beta$ implies $\vdash \Gamma \Rightarrow \mathrm{X}^i!\beta$ for any $\Gamma \in M$. Suppose $(\Gamma, i) \models !\beta$. Then there exists $\Delta \in M$ such that $\vdash \Gamma \Rightarrow (!\Delta)^\star$ and $(\Delta, i) \models \beta$. By the hypothesis of induction, we obtain $\vdash \Delta \Rightarrow \mathrm{X}^i\beta$. Thus we obtain $\vdash \Gamma \Rightarrow \mathrm{X}^i!\beta$:

$$\dfrac{\dfrac{\dfrac{\Delta \Rightarrow \mathrm{X}^i\beta}{\vdots\ (!\text{left})}}{!\Delta \Rightarrow \mathrm{X}^i\beta}\ (!\text{right})}{\vdots\ (*\text{left})\ \text{or}\ (\mathbf{1}\text{we})} \qquad \dfrac{\Gamma \Rightarrow (!\Delta)^\star \quad (!\Delta)^\star \Rightarrow \mathrm{X}^i!\beta}{\Gamma \Rightarrow \mathrm{X}^i!\beta}\ (\text{cut}).$$

Conversely, suppose $\vdash \Gamma \Rightarrow \mathrm{X}^i!\beta$. We will show $(\Gamma, i) \models !\beta$, i.e. there exists $\Delta \in M$ such that $\vdash \Gamma \Rightarrow (!\Delta)^\star$ and $(\Delta, i) \models \beta$. We take $\{\mathrm{X}^i\beta\}$ for Δ. Then we obtain $(\{\mathrm{X}^i\beta\}, i) \models \beta$ by the induction hypothesis. Using the hypothesis $\vdash \Gamma \Rightarrow \mathrm{X}^i!\beta$, we obtain $\vdash \Gamma \Rightarrow !\mathrm{X}^i\beta$:

$$\dfrac{\Gamma \Rightarrow \mathrm{X}^i!\beta \quad \dfrac{\dfrac{\mathrm{X}^i\beta \Rightarrow \mathrm{X}^i\beta}{\mathrm{X}^i!\beta \Rightarrow \mathrm{X}^i\beta}\ (!\text{left})}{\mathrm{X}^i!\beta \Rightarrow !\mathrm{X}^i\beta}\ (!\text{right})}{\Gamma \Rightarrow !\mathrm{X}^i\beta}\ (\text{cut}).$$

Thus, we obtain $\vdash \Gamma \Rightarrow (!\Delta)^\star$.
(Case $\gamma \equiv \mathrm{X}\alpha$): $(\Gamma, i) \models \mathrm{X}\alpha$ iff $(\Gamma, i+1) \models \alpha$ iff $\vdash \Gamma \Rightarrow \mathrm{X}^{i+1}\alpha$ (by the induction hypothesis) iff $\vdash \Gamma \Rightarrow \mathrm{X}^i\mathrm{X}\alpha$.
(Case $\gamma \equiv \mathrm{G}\alpha$): Suppose $(\Gamma, i) \models \mathrm{G}\alpha$. Then we have $\forall l \geq i\ [(\Gamma, l) \models \alpha]$, and hence $\forall l \geq i\ [\vdash \Gamma \Rightarrow \mathrm{X}^l\alpha]$ by the induction hypothesis. This means $\forall k \in \omega\ [\vdash \Gamma \Rightarrow \mathrm{X}^{i+k}\alpha]$, and thus $\vdash \Gamma \Rightarrow \mathrm{X}^i\mathrm{G}\alpha$ by (Gright). Conversely, suppose $\vdash \Gamma \Rightarrow \mathrm{X}^i\mathrm{G}\alpha$. Then we have:

$$\dfrac{\Gamma \Rightarrow \mathrm{X}^i\mathrm{G}\alpha \quad \dfrac{\mathrm{X}^{i+k}\alpha \Rightarrow \mathrm{X}^{i+k}\alpha}{\mathrm{X}^i\mathrm{G}\alpha \Rightarrow \mathrm{X}^{i+k}\alpha}\ (\text{Gleft})}{\Gamma \Rightarrow \mathrm{X}^{i+k}\alpha}\ (\text{cut})$$

for any $k \in \omega$, i.e. $\forall l \geq i\ [\vdash \Gamma \Rightarrow \mathrm{X}^l\alpha]$. By the hypothesis of induction, we obtain $\forall l \geq i\ [(\Gamma, l) \models \alpha]$ and hence $(\Gamma, i) \models \mathrm{G}\alpha$. Q.E.D.

Lemma 15 *The canonical model* $\langle M, N, \cdot, \dagger, \varepsilon, \geq, \models \rangle$ *defined in Definition 13 is a Kripke model for* LT *such that*

$$(\varepsilon, 0) \models \gamma \textit{ if and only if } \mathrm{LT} \vdash \Rightarrow \gamma$$

for any formula γ.

Proof The hereditary condition on $\models$ is obvious. By taking 0 for i and taking ε for Γ in Lemma 14, the required fact is obtained. Q.E.D.

By using Lemma 15, the following theorem is obtained, because for any sequent $\Delta \Rightarrow \delta$, it can take the formula $\Delta^\star \rightarrow \delta$ such that $\vdash \Delta \Rightarrow \delta$ iff $\vdash \Rightarrow \Delta^\star \rightarrow \delta$.

Theorem 16 (Completeness) Let C be a class of Kripke frames for LT, $L := \{S \mid \mathrm{LT} \vdash S\}$ and $L(C) := \{S \mid S \textit{ is valid in all frames of } C\}$. Then $L(C) \subseteq L$.

4. Timed Petri net interpretation

The following definition of timed Petri net is roughly the same as that in (Tanabe, 1997).

Definition 17 (Timed Petri net) *A timed Petri net is a structure* $\langle N, P, T, (\cdot)^\bullet, (\cdot)_\bullet \rangle$ *such that*

8. *N* is the set of natural numbers representing liner-time,
9. *P is a set of places,*
10. *T is a set of transitions,*
11. $(\cdot)^\bullet$ *and* $(\cdot)_\bullet$ *are mappings from T to the set S of all multisets over P* × *N.*

For $t \in T$, $t^\bullet$ *and* $t_\bullet$ *are called the* pre-multiset *and the* post-multiset *of t respectively. Each element of S is called a* timed marking.

In this definition, $i\ (\in N)$ indicates the waiting time until the pending tokens which are usable in future become available in a place. Thus, an expression $(\alpha, i)\ (\in P \times N)$, which corresponds to the formula $\mathrm{X}^i\alpha$, means "A token α has pending time i, i.e., α will be active after *i* time units." In such an expression, a token $(\alpha, 0)$ is called an *active token*, and a token (α, i) with $i \neq 0$ is called a *pending token*.

Definition 18 (Reachability relation) A firing relation $[t\rangle$ *for* $t \in T$ *on* S is defined as follows: for any m_1, $m_2 \in$ S,

$$m_1\ [t\rangle\ m_2 \quad \textit{iff} \quad m_1 = m_3 + t^\bullet \ \textit{and}\ t_\bullet + m_3 = m_2 \ \textit{for some}\ m_3 \in S.$$

A reachability relation $\gg$ *on S is defined as follows: for any* $m, m' \in S$,
$m \gg m'$ *iff* $m\ [t_1\rangle\ m_1\ [t_2\rangle \cdots [t_n\rangle\ m_n = m'$ *for some* $t_1, ..., t_n \in T$, $m_1, ..., m_n \in S$ *and* $n \geq 0$.
It is remarked that $\gg$ *is transitive and reflrxive.*

We sometimes have to add certain time passage functions and timing conditions to the definitions of timed Petri net, firing relation and reachability relation, in case-by-case. A time passage function δ_i, which means the passage of time by *i* time units, is a function on S such that $\delta_i(\{(\alpha_j, k)\}_{j \in I}) = \{(\alpha_j, k+i)\}_{j \in I}$. A firing relation may be extended with respect to such time passage functions such that $m\ [\delta_i\rangle\ m'$ iff $\delta_i(m) = m'$. A timing condition TC is a binary relation on S. Then an extended reachability relation $\gg$ on S may have, for example, the following conditions:

1. $\forall m, m' \in S\ [\delta_i(m) = m'$ implies $m \gg m']$,
2. $\forall m, m' \in S\ [\langle m, m' \rangle \in \mathrm{TC}\ \textit{implies}\ m \gg m']$,
3. $\forall m, m', m'' \in S\ [m \gg m'$ implies $(m+m'' \gg m'+m''$ and $\delta_i(m) \gg \delta_i(m'))]$

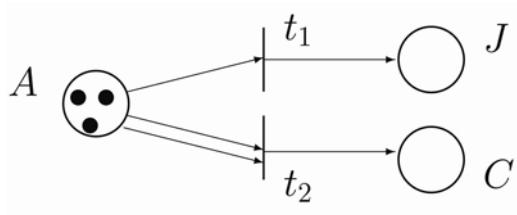
Following (Tanabe, 1997), we give an example of timed Petri nets.

Example 19 (Apple drinks 1) Suppose that we have just picked up three apples from an apple tree, and we can choose apple drinks between two options according to the following two rules.

(Rule 1): *from an apple of less than one month old (i.e., less than a month has passed since picked from the tree), we can make a glass of apple juice.*

(Rule2): *from two apples of between 10 and 20 months old, we can make a glass of cider.*

We then give a timed Petri net $\langle N, P, T, (\cdot)^\bullet, (\cdot)_\bullet \rangle$ *with two time passage functions* δ_1 *and* δ_{11}, *and two timing conditions* TC1 and TC2. *Let P be {A, J, C} where A, J and C correspond to an apple, a glass of juice and a glass of cider, respectively. Let T be {t1,t2},* $t_1{}^\bullet$ *be {(A, i)},* $t_{1\bullet}$ be {(J, i)}, $t_2{}^\bullet$ be *{(A, i), (A, i))}, and* $t_{2\bullet}$ *be {(C, i)}. Let* TC1 *be {(A, x)}*$\gg$ *{(J, 0)}* $(0 \leq x \leq 1)$, *and* TC2 *be* $\{(A, x_1), (A, x_2)\} \gg \{(C, 0)\}$ $(10 \leq x_i \leq 20, i = 1, 2)$. *It is remarked that* TC1 *and* TC2 *correspond to* (Rule1) *and* (Rule2), *respectively. Graphically this becomes the following*:



In this net, "•" indicates a timed token (A, i).

Example 20 (Apple drinks 2) In Example 19, we consider the situation (Tanabe, 1997) that starting from three apples, how can we get a glass of juice and a glass of cider? Going through the stage of getting drinks.

1. *We have three fresh apples: {(A, 0), (A, 0), (A, 0)}.*
2. *One month has passed, i.e., all the apples has become one month old:*
3. *{(A, 1), (A, 1), (A, 1)}.*
4. *A glass of juice is made from an apple: {(J, 0), (A, 1), (A, 1)}.*
5. *More eleven months have passed: {(J, 11), (A, 12), (A, 12)}.*
6. *We finally have a glass of juice and a glass of cider: {(J, 11), (C, 0)}.*

Then this situation is expressed as follows:

$$\{(A,0),(A,0),(A,0)\} \quad [\delta_1\rangle$$
$$\{(A,1),(A,1),(A,1)\} \quad [t_1\rangle$$
$$\{(J,0),(A,1),(A,1)\} \quad [\delta_{11}\rangle$$
$$\{(J,11),(A,12),(A,12)\} \quad [t_2\rangle$$
$$\{(J,11),(C,0)\}.$$

Thus we can obtain:

$$\{(A,0),(A,0),(A,0)\} \quad \gg \quad \{(J,11),(C,0)\}$$

In the next example, this will be verified using LT.

In order to compare timed Petri net and LT, the following definition is considered. It is assumed here that there is no time passage function or timing condition, since these are additional items in case-by-case.

Definition 21 (Timed Petri net structure) *A timed Petri net structure is a structure* $\langle S, N, +, \emptyset, \gg\rangle$ such that

7. *N is the set of natural numbers representing liner-time,*
8. *S is the set of all timed-markings,*
9. *+ is a multiset union operation on S,*
10. *$\emptyset$ is the empty multiset,*
11. *$\gg$ is a reachability relation on S.*

It is remarked that a timed Petri net structure $\langle S, N, +, \emptyset, \gg\rangle$ satisfies the following conditions:

1. $\langle S, +, \emptyset\rangle$ is a commutative monoid,
2. $\langle S, \gg\rangle$ is a pre-ordered set,
3. $x_1 \gg x_2$ and $y_1 \gg y_2$ imply $x_1 + y_1 \gg x_2 + y_2$ for all $x_1, x_2, y_1, y_2 \in S$.

We then have the following basic proposition.

Proposition 22 (Correspondence: Timed Petri net and Kripke frame)

A timed Petri net structure $\langle S, N, +, \emptyset, \gg\rangle$ is just a †-free reduct of a Kripke frame for LT.

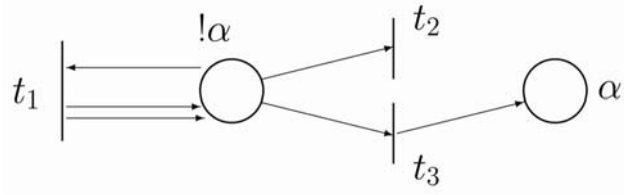
By this prposition and the canonical model defined in Definition 13, a timed Petri net interpretation for LT is obtained.

1. A timed token or place name, (α, i) or $(\alpha, 0)$, corresponds to the formula $\mathrm{X}^i\alpha$ or α.

2. The reachability of a timed Petri net corresponds to the provability of a sequent in LT, i.e. $\Gamma \gg \Delta$ corresponds to $\mathrm{LT} \vdash \Gamma \Rightarrow \Delta^*$.

Then we have the remained question: "What is the Petri net interpretation of the exponential operator?" The following example is an answer from the idea of Ishihara and Hiraishi (Ishihara & Hiraishi, 2001).

Example 23 (Exponential operator) *We give a timed Petri net* $\langle N, P, T, (\cdot)^\bullet, (\cdot)_\bullet \rangle$ *with* $P := \{!\alpha, \alpha\}$, $T := \{t_1, t_2, t_3\}$, ${t_1}^\bullet := \{(!\alpha, 0)\}$, ${t_1}_\bullet := \{(!\alpha, 0)\}$, ${t_2}^\bullet := \{(!\alpha, 0)\}$, ${t_2}_\bullet := \{\}$, ${t_3}^\bullet := \{(!\alpha, 0)\}$ *and* ${t_3}_\bullet := \{(\alpha, 0)\}$, *where all tokens are active tokens, i.e., tokens with* $i = 0 \in N$. *Graphically this becomes the following:*



This net corresponds to the facts $\vdash !\alpha \Rightarrow !\alpha * !\alpha$, $\vdash !\alpha \Rightarrow \mathbf{1}$ *and* $\vdash !\alpha \Rightarrow \alpha$. *In this net, the place* $!\alpha$ *(if it has a timed token) can produce a number of tokens in many-times (i.e. as many as needed).*

We now show a LT based expression of the timed Petri net in the apple drink examples discussed before.

Example 24 (Apple drinks 3) We recondider Examples 19 and 20 based on a sequent calculus expression for LT.

The time passage functions δ_1 *and* δ_{11}, *and the timing conditions TC1 and TC2 are expressed as initial sequents (non-logical axioms) for* LT.

δ_j $(j \in \{1, 11\})$: $\vdash \mathrm{X}^i\alpha, \mathrm{X}^i\beta, \mathrm{X}^i\gamma \Rightarrow \mathrm{X}^{i+j}\alpha, \mathrm{X}^{i+j}\beta, \mathrm{X}^{i+j}\gamma$ *for any formulas* α, β *and* γ.

TC1: $\vdash \mathrm{X}^i A \Rightarrow J \quad (0 \leq i \leq 1)$.

TC2: $\vdash \mathrm{X}^i A, \mathrm{X}^i A \Rightarrow C \quad (10 \leq i\ (j) \leq 20)$.

In the following, we verify $A, A, A \Rightarrow \mathrm{X}^{11} J * C$, *i.e.,* $\{(A, 0), (A, 0), (A, 0)\} \gg \{(J, 11), (C, 0)\}$.

$$\dfrac{\dfrac{\vdots\ \delta_1}{A, A, A \Rightarrow \mathrm{X}A * \mathrm{X}A * \mathrm{X}A} \qquad \dfrac{\dfrac{\dfrac{\vdots\ \mathrm{TC1} \quad \dfrac{\mathrm{X}A \Rightarrow \mathrm{X}A \quad \mathrm{X}A \Rightarrow \mathrm{X}A}{\mathrm{X}A, \mathrm{X}A \Rightarrow \mathrm{X}A * \mathrm{X}A}}{\mathrm{X}A, \mathrm{X}A, \mathrm{X}A \Rightarrow J * \mathrm{X}A * \mathrm{X}A} \qquad \vdots}{J * \mathrm{X}A * \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C}}{\mathrm{X}A, \mathrm{X}A, \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C}}{\mathrm{X}A * \mathrm{X}A * \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C}}{A, A, A \Rightarrow \mathrm{X}^{11} J * C}$$

where $J * \mathrm{X}A * \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C$ *is proved by*

$$\dfrac{\dfrac{\vdots\ \delta_{11}}{J, \mathrm{X}A, \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * \mathrm{X}^{12} A * \mathrm{X}^{12} A} \qquad \dfrac{\dfrac{\vdots\ \mathrm{TC2}}{\mathrm{X}^{11} J \Rightarrow \mathrm{X}^{11} J \quad \mathrm{X}^{12} A * \mathrm{X}^{12} A \Rightarrow C}}{\mathrm{X}^{11} J, \mathrm{X}^{12} A * \mathrm{X}^{12} A \Rightarrow C}}{\mathrm{X}^{11} J * \mathrm{X}^{12} A * \mathrm{X}^{12} A \Rightarrow \mathrm{X}^{11} J * C}}{J, \mathrm{X}A, \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C}}{J * \mathrm{X}A * \mathrm{X}A \Rightarrow \mathrm{X}^{11} J * C}\ .$$

5. Concluding remarks

In this paper, a new logic, called linear-time linear logic, was introduced as two equivalent cut-free sequent calculi LT and 2LT, which are the linear logic versions of Kawai's LT_ω and Baretella and Masini's 2Sω for the standard linear-time temporal logic. The completeness theorem w.r.t. the Kripke semantics with a natural timed Petri net interpretation was proved for LT as the main result of this paper. By using this theorem, a relationship between LT and a timed Petri net with timed tokens was clarified, and the reachability of such a Petri net was transformed into the provability of LT and also 2LT. This means that the timed Petri net can naturally be expressed as the proof-theoretic framework by LT.

In the following, some technical remarks are given. The Kripke semantics presented is similar to the Kripke semantics (or resource algebras) with location interpretations by Kobayashi, Shimizu and Yonezawa (Kobayashi et al., 1999)) and Kamide (Kamide, 2004). The sequent calculi and Kripke semantics for LT can also be adapted to Lafont's (intuitionistic) soft linear logic (Lafont, 2004) by using the framework presented in (Kamide, 2004). The framework posed in this paper can be extended to a rich framework with the first-order universal quantifier $\forall$,based on the technique posed in (Kamide, 2004). It is known in (Lilius, 1992) that the linear logic framework with the first-order quantifiers correspond to a high-level Petri net framework.

6. References

Baratella, S. and Masini, A. (2004). An approach to infinitary temporal proof theory. *Archive for Mathematical Logic* 43 (8), pp. 965–990.

Bestuzheva, I.I. & Rudnev, V.V. (1994). Timed Petri nets: Classification and comparative analysis. *Automation and Remote Control* 51 (10), pp. 1308–1318.

Clarke, E.M., Grumberg, O. & Peled, D. A. (1999). Model checking. The MIT Press.

Došen, K. (1988). Sequent systems and groupoid models I, II. *Studia Logica* 47, 48, pp. 353–385, pp. 41–65.

Emerson, E.A. (1990). Temporal and modal logic. In Handbook of Theoretical Computer Science, Formal Models and Semantics (B), Jan van Leeuwen (Ed.), pp. 995–1072, Elsevier and MIT Press.

Engberg, U. & Winskel, G. (1997). Completeness results for linear logic on Petri nets. *Annals of Pure and Applied Logic* 86, pp. 101–135.

Farwer, B. (1999). A linear logic view of object Petri nets. *Fundamenta Informaticae* 37 (3), pp. 225–246.

Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science* 50, pp. 1–102.

Hirai, T. (2000). Propositional temporal linear logic and its application to concurrent systems. *IEICE Transactions (Fundamentals)* E83-A 11, pp. 2219–2227.

Hirai, T. (1999). An application of a temporal linear logic to timed Petri nets. Proceedings of Petri nets' 99, Workshop on applications of Petri nets to intelligent system development, pp. 2–13.

Hodas, J. & Miller, D. (1994). Logic programming in a fragment of intuitionistic linear logic. *Information and Computation* 110, pp. 327–365.

Ishihara, K. & Hiraishi, K. (2001). The completeness of linear logic for Petri net models. *Logic Journal of the IGPL 9*, No. 4, pp. 549–567.

Kamide, N. (2002). Kripke semantics for modal substructural logics. *Journal of Logic, Language and Information* 11, pp. 453–470.

Kamide, N. (2003). A simplified semantics for a fragment of intuitionistic linear logic. *Bulletin of the Section of the Logic* 32 (3), pp. 121–127.

Kamide, N. (2004). Combining soft linear logic and spatio-temporal operators. Journal of Logic and Computation 14 (5), pp. 625–650, 2004.

Kamide, N. (2006a). Linear and affine logics with temporal, spatial and epistemic operators. Theoretical Computer Science 353 (1–3), pp. 165–207.

Kamide, N. (2006b). An equivalence between sequent calculi for linear-time temporal logic. *Bulletin of the Section of the Logic* 35 (4), pp. 187–194.

Kamide, N. (2006c). Phase semantics and Petri net intepretation for resource-sensitive strong negation. *Journal of Logic, Language and Information* 15 (4), pp. 371–401.

Kamide, N. (2007). Phase semantics for linear-time formalism. Preprint 2007.

Kanovich, M.I. & Ito, T. (1998). Temporal linear logic specifications for concurrent procesess (extended abstract). Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, pp. 48–57.

Kanovich, M.I. (1995). Petri nets, Horn programs, linear logic and vector games. *Annals of Pure and Applied Logic* 75, pp. 107–135.

Kanovich, M.I. (1994). Linear logic as a logic of computations. *Annals of Pure and Applied Logic* 67, pp. 183–212.

Kawai, H. (1987). Sequential calculus for a first order infinitary temporal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 33, pp. 423–432.

Kosaraju, S. (1973). Limitations of Dijkstra's semaphore primitives and Petri nets. *Operating Systems Review* 7, No. 4, pp. 122–126.

Kobayashi, N., Shimizu, T. & Yonezawa, A. (1999). Distributed concurrent linear logic programming. *Theoretical Computer Science* 227, pp. 185–220.

Kröger, F. (1977). LAR: a logic of algorithmic reasoning. *Acta Informatica* 8, pp. 243–266.

Lafont, Y. (2004). Soft linear logic and polynomial time. *Theoretical Computer Science* 318 (1-2), pp. 163–180.

Larchey-Wendling, D & Galmiche, D. (1998). Provability in intuitionistic linear logic from a new interpretation on Petri nets — extended abstract —. *Electronic Notes in Theoretical Computer Science* 17, 18 pages, 1998.

Larchey-Wendling, D. & Galmiche, D. (2000). Quantales as completions of ordered monoids: revised semantics for intuitionistic linear logic. *Electronic Notes in Theoretical Computer Science* 35, 15 pages.

Lichtenstein, O. & Pnueli, A. (2000). Propositional temporal logics: decidability and completeness. *Logic Journal of the IGPL 8* (1), pp. 55–85.

Lilius, J. (1992). High-level nets and linear logic. Lecture Notes in Computer Science 616, Springer-Verlag, pp. 310–327.

Martí-Oliet, N. & Meseguer, J. (1991). From Petri nets to linear logic. *Mathematical*

Structures in Computer Science 1, pp. 69–101.

Okada, M. (1998). An introduction to linear logic: expressiveness and phase semantics. *MSJ Memoirs* 2, pp. 255–295.

Ono, H. & Komori, Y. (1985). Logics without the contraction rule. *Journal of Symbolic Logic* 50, pp. 169–201.

Paech, B. (1988). Gentzen-systems for propositional temporal logics. Lecture Notes in Computer Science 385, pp. 240–253.

Pliuškevičius, R. (1991). Investigation of finitary calculus for a discrete linear time logic by means of infinitary calculus. Lecture Notes in Computer Science 502, pp. 504–528.

Pnueli, A. (1977). The temporal logic of programs. Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–57.

Reisig, W & Rozenberg, G. (Eds.), (1998a). Lectures on Petri nets I: basic models (Advances in Petri nets). Lecture Notes in Computer Science 1491, Springer-Verlag.

Reisig, W. & Rozenberg, G. (Eds.), (1998b). Lectures on Petri nets II: applications (Advances in Petri nets). Lecture Notes in Computer Science 1492, Springer-Verlag.

Szabo, M.E. (1980). A sequent calculus for Kröger logic. Lecture Notes in Computer Science 148, pp. 295–303.

Szałas, A. (1986). Concerning the semantic consequence relation in first-order temporal logic. *Theoretical Computer Science* 47 (3), pp. 329–334.

Tanabe, M. (1997). Timed Petri nets and temporal linear logic. Lecture Notes in Computer Science 1248, pp. 156–174.

Tamura, N., Hirai, T., Yoshikawa, H., Kan, Kyoung-San & Banbara, M. (2000). Logic programming in an intuitionistic temporal linear logic. *Information Processing Society of Japan, Transactions on Programming* 41, SIG4 (PRO 7), pp. 11–23 (in japanease).

Troelstra, A.S. (1992). Lectures on linear logic. CSLI Lecture Notes, Vol. 29, Stanford, CA: CSLI.

Urquhart, A. (1972). Semantics for relevant logics. *Journal of Symbolic Logic* 37, pp. 159–169.

Vardi, M.Y. (2001). Branching vs. linear-time: final showdown. Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), pp. 1–22.

Vardi, M.Y. (2007). Automata-theoretic model checking revisited. Invited paper of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07), 14 pages.

Wang, J. (1998). Timed Petri nets: theory and application. Kluwer Academic Publishers, Boston.

Wansing, H. (1993a). Informational interpretation of substructural propositional logics. *Journal of Logic, Language and Information* 2, pp. 285–308.

Wansing, H. (1993b). The logic of information structures. Lecture Notes in Artificial Intelligence 681, pp. 1–163.

11

From Time Petri Nets to Timed Automata

Franck Cassez and Olivier H. Roux
Institut de Recherche en Communication et Cybernétique de Nantes (IRCCyN)
France

1. Introduction

In this chapter we introduce a formalism, *Time Petri Nets (TPNs)*, to model real-time systems. We compare it with another well-known formalism, *Timed Automata (TA)*, used for specifying timed systems. We precisely define the semantics of TPNs and TA and compare them according to two criteria: the *languages* (or set of behaviours) they can generate, and the *trees* (or branching behaviours) they can generate. We show that every TPN can be translated into an equivalent[1] TA.
Then, we introduce a real-time logic to specify properties of real-time systems. We show how to check that a given TPN satisfies a property written in this logic. For this, we use our translation[2] from TPNs to TA and check the property on the equivalent TA. Finally we briefly report on experiments for checking real-time properties of TPNs using this framework.

1.1. Petri Nets with Time

The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) (Merlin, 1974) and Timed Petri Nets (Ramchandani, 1974). In a TPN a transition can fire within a time interval whereas for Timed Petri Nets it fires as soon as possible. For Timed Petri Nets, time can be considered relative to places or transitions (Sifakis, 1980; Pezzè, 1999). It is interesting to formally compare the different classes of Petri Nets with time: this gives a better idea of what one subclass should be used for. The expressive power of (time) Petri Nets can be compared w.r.t. the set of (timed) behaviors they can generate. One class C is a subclass of another C', if for every net n in C, there is a net n' in C' which can generate the same behaviors as n. In this case, we say that the class C is less expressive than C'. For instance, the two subclasses P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent (Sifakis, 1980; Pezzè, 1999) (i.e., P-Timed Petri Nets are less expressive than T-Timed Petri Nets and vice-versa). The same subclasses are defined for TPNs i.e., T-TPNs and P-TPNs. Both classes of Timed Petri Nets are less expressive than both P-TPNs and T-TPNs (Pezzè, 1999). P-TPNs and T-TPNs are incomparable (Khansa et al., 1996). Finally TPNs are less expressive than Time Stream Petri Nets (Diaz and Senac, 1994) which were introduced to model multimedia applications.
Another way of comparing two classes is to determine the status of different decision problems (e.g., *reachability*, *coverability*, *boundedness*) for the two classes. For instance, reachability is undecidable for TPNs, as well as boundedness. Recent work (de Frutos Escrig et al., 2000; Abdulla and Nylén, 2001) considers timed arc Petri nets where each token has a clock representing its "age". The authors prove that coverability and boundedness are decidable for this class of Petri nets by applying a backward exploration technique. They use a lazy (non-urgent) behavior of the net: the firing of transitions may be delayed, even if that implies that

[1] This equivalence is formally defined in the chapter.
[2] This translation preserves the properties of this logic.

some transitions are disabled because their input tokens become too old.
The class T-TPNs is the most commonly-used subclass of TPNs to specify real-time systems. In this chapter, we focus on this subclass that will be henceforth referred to as TPNs. For classical TPNs (with closed intervals), boundedness is undecidable (Berthomieu and Diaz, 1991), and papers on this model report undecidability results, or decidability under the assumption that the TPN is bounded, e.g., reachability in (Popova, 1991).

1.2. Verifying Time Petri Nets

The main objective in specifying real-time systems with formalisms like TPNs is to build a model of a system S, and be able to mathematically reason about it. By reasoning we mean "verifying that some properties are satisfied on the model". The properties we would like to check range from simple ones like "the system cannot reach a bad state" which are *reachability properties*, to more involved properties like "after each failure, the system will reach a stable state within 10 time units", which are *quantitative real-time* properties. Given a formal model S and a temporal logic formula φ, verifying that S satisfies φ is usually achieved by a *model-checking* algorithm: such an algorithm checks that S is a model of the formula φ. Hence the process of verifying that a formal model of a system satisfies a property in a temporal logic is often called *model-checking*.
Algorithms for verifying properties on TPNs have been designed for more than a decade. Formally, the *behavior* of a TPN can be defined by timed firing sequences which are sequences of pairs (t, d) where t is a transition of the TPN and $d \in \mathbb{R}_{\geq 0}$. A sequence of transitions like $\omega = (t_1, d_1)\ (t_2, d_2) \ldots (t_n, d_n) \ldots$ indicates that t_1 is fired after d_1 time units, then t_2 is fired after d_2 time units have elapsed since t_1 was fired, and so on, so that transition t_i is fired at absolute time $\sum_{k=1}^{i} d_k$. A *marking* M is *reachable* in a TPN if there is a timed firing sequence ω from the initial marking M_0 to M. Reachability analysis of TPNs relies on the construction of the so-called State-Class Graph (SCG) that was introduced in (Berthomieu and Menasche, 1983) and later refined in (Berthomieu and Diaz, 1991). It has been recently improved in (Lilius, 1998) by using partial-order reduction methods.
For bounded TPNs, the SCG construction obviously solves the *marking reachability* problem: "Given a marking M, is it possible to reach M from M_0?". If one wants to solve the *state reachability* problem: "Given M and $v \in \mathbb{R}_{\geq 0}$ and a transition t, can we reach a marking M such that transition t has been enabled for v time units?", the SCG is not precise enough and an alternative graph, the *Strong State Class Graph* has to be built for this purpose (Berthomieu and Vernadat, 2003). The previous two graphs allow for checking *qualitative* properties written in a real-time logic called LTL (Emerson, 1990). A more powerful real-time logic, CTL* (Emerson, 1990), can be checked on TPNs using yet another more precise graph.
Anyway, none of the previous graphs is a good[3] abstraction (accurate enough) for checking *quantitative* real-time properties e.g., "it is not possible to stay in marking M more than n time units" or "from marking M, marking M' is always reached within n time units". In this chapter we introduce a logic to specify such quantitative real-time properties and present an algorithm to check for such properties on TPNs.

1.3. Timed Automata

Timed Automata (TA) were introduced by Alur & Dill (Alur and Dill, 1994) and have since been extensively studied. They are now widely used to model real-time systems. TA forms an extension of finite automata with dense time *clocks* and enables one to specify real-time

[3]The use of *observers* is of little help as it requires to specify a property as a TPN; thus it is hard to specify properties on markings.

systems. It has been shown that model-checking a quantitative real-time logic, called TCTL, is decidable (Alur and Dill, 1994; Henzinger et al., 1994) for TA and some of their extensions (Bouyer et al., 2000). There also exist several efficient tools like UPPAAL (Larsen et al., 1997), KRONOS (Yovine, 1997) and CMC (Laroussinie and Larsen, 1998) for model-checking TA and many real-time industrial applications have been specified and successfully verified with them. In this chapter, we show how to translate TPNs into equivalent TA. This enables us to use the technology and tools developed for TA to verify TPNs.

1.4. Outline of the Chapter

In Section 2 we fix notations and provide basic notions for defining the formal semantics of Time Petri Nets and Timed Automata. In Section 3, we introduce Time Petri Nets and define their semantics. We also give the main properties of this model together with an algorithm to compute a finite representation of the state space of a Time Petri Net; this algorithm can be used to check marking reachability. In Section 4, we compare the expressiveness of Timed Automata and Time Petri Nets and show that they are equivalent w.r.t. language equivalence. We give the translation from TPNs to TA and show in Section 5 how to check quantitative properties on TPNs using a timed temporal logic. In Section 6, we apply the framework defined in Section 4 on some examples. Finally, we conclude with recent or ongoing work on this subject in Section 7.

2. Preliminaries

Let Σ be a finite alphabet. Σ^* (resp. Σ^ω) denotes the set of finite (resp. infinite) sequences over Σ and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ (called words in the sequel). By convention if $u \in \Sigma^\omega$, then the *length* of u, denoted $|u|$, is ω; otherwise if $u = a_1 \cdots a_n$, $|u| = n$. We also use $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ where $\varepsilon \notin \Sigma$, where ε is the empty word. B^A stands for the set of mappings from A to B. If A is finite and $|A| = n$, an element of B^A is also a vector in B^n. The usual operators $+, -, <$ and $=$ are used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ and are the point-wise extensions of their counterparts in A. The set $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ denotes the boolean truth values *true* and *false*, $\mathbb{R}_{\geq 0}$ denotes the set of non-negative reals and $\mathbb{R}_{>0} = \mathbb{R}_{\geq 0} \setminus \{0\}$. A *valuation* ν for the set of variables X is an element of $\mathbb{R}_{\geq 0}^X$. For $\nu \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ denotes the valuation defined by $(\nu + d)(x) = \nu(x) + d$, and for $X' \subseteq X$, $\nu[X' \mapsto 0]$ denotes the valuation ν' with $\nu'(x) = 0$ for $x \in X'$ and $\nu'(x) = \nu(x)$ otherwise. $\mathbf{0}$ denotes the valuation s.t. $\forall x \in X, \mathbf{0}(x) = 0$. An *atomic constraint* is a formula of the form $x \bowtie c$ for $x \in X$, $c \in \mathbb{Q}_{\geq 0}$ and $\bowtie \in \{<, \leq, \geq, >\}$. We denote $\mathcal{C}(X)$ the set of *constraints* over a set of variables X which consists of conjunctions of atomic constraints. Given a constraint $\varphi \in \mathcal{C}(X)$ and a valuation $\nu \in \mathbb{R}_{\geq 0}^X$, we denote $\varphi(\nu) \in \mathbb{B}$ the truth value obtained by substituting each occurrence of x in φ by $\nu(x)$.

2.1. Timed Languages and Timed Transition Systems

Let Σ be a fixed finite alphabet s.t. $\varepsilon \notin \Sigma$ and A be a finite alphabet which can contain ε.

Definition 1 (Timed Word) *A* timed word *w over Σ is a finite or infinite sequence*

$$w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$$

s.t. for each $i \geq 0$, $a_i \in \Sigma$, $d_i \in \mathbb{R}_{\geq 0}$ and $d_{i+1} \geq d_i$.

A timed word $w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$ over Σ can also be viewed as a pair $(v, \tau) \in \Sigma^\infty \times \mathbb{R}_{\geq 0}^\infty$ s.t. $|v| = |\tau|$. The value d_k gives the absolute time (considering the initial instant is 0) of action a_k. We write $\mathit{Untimed}(w) = a_0 a_1 \cdots a_n \cdots$ for the untimed part of w, and

Duration$(w) = \sum_{k\geq 0} d_k$ for the duration of the timed word w. We let $\mathrm{TW}^*(\Sigma)$ (resp. $\mathrm{TW}^\omega(\Sigma)$) be the set of finite (resp. infinite) timed words over Σ and $\mathrm{TW}(\Sigma) = \mathrm{TW}^*(\Sigma) \cup \mathrm{TW}^\omega(\Sigma)$. A *timed language* L over Σ is a set of timed words i.e., any set $L \subseteq \mathrm{TW}(\Sigma)$.
Timed Transition Systems (TTS) are usual transition systems with two types of labels: discrete labels for events and positive reals' labels for time elapsing. Consequently, they have two types of transitions: discrete transitions and time transitions:

Definition 2 (Timed Transition System) *A* timed transition system *(TTS) (over a set of actions A) is a tuple* $S = (Q, Q_0, A, \to, F, R)$ *where:*

- Q *is a set of states;*
- $Q_0 \subseteq Q$ *is the set of initial states;*
- A *is a finite set of actions disjoint from* $\mathbb{R}_{\geq 0}$*;*
- $\longrightarrow \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ *is a set of edges. If* $(q, e, q') \in \longrightarrow$*, we also write* $q \xrightarrow{e} q'$*;*
- $F \subseteq Q$ *and* $R \subseteq Q$ *are respectively the set of* final *and* repeated *states.*

For a time transition $q \xrightarrow{d} q'$ with $d \in \mathbb{R}_{\geq 0}$, d denotes a delay and not an absolute time. We assume that in any TTS there is a transition $q \xrightarrow{0} q'$ and in this case $q = q'$. A *run* ρ of length $n \geq 0$ is a finite ($n < \omega$) or infinite ($n = \omega$) sequence of alternating time and discrete transitions of the form:

$$\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_0} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_1} \cdots q_n \xrightarrow{d_n} q'_n \cdots$$

with $d_i \in \mathbb{R}_{\geq 0}$ and $a_i \in A$. We write $\mathit{first}(\rho) = q_0$. We assume that a finite run ends with a time transition d_n. If ρ ends with d_n, we let $\mathit{last}(\rho) = q'_n$ and write $q_0 \xrightarrow{d_0 a_0 \cdots d_n} q'_n$. We write $q \xrightarrow{*} q'$ if there is run ρ s.t. $\mathit{first}(\rho) = q_0$ and $\mathit{last}(\rho) = q'$. The set of *reachable states* in S is the set of states q s.t. $q_0 \xrightarrow{*} q$ for some $q_0 \in Q_0$. The *trace* of an infinite run ρ is the timed word $\mathit{trace}(\rho) = (a_{i_0}, d_0 + \cdots + d_{i_0}) \cdots (a_{i_k}, d_0 + \cdots + d_{i_k}) \cdots$ that consists of the sequence of letters $(a_{i_k})_{k\in\mathbb{N}}$ of $A \setminus \{\varepsilon\}$. If ρ is a finite run, we define the trace of ρ by $\mathit{trace}(\rho) = (a_{i_0}, d_0 + \cdots + d_{i_0}) \cdots (a_{i_k}, d_0 + \cdots + d_{i_k})$ where the a_{i_k} are in $A \setminus \{\varepsilon\}$. By definition $\mathit{Untimed}(\rho) = \mathit{Untimed}(\mathit{trace}(\rho))$ and $\mathit{Duration}(\rho) = \sum_{d_k \in \mathbb{R}_{\geq 0}} d_k$. A run ρ is *zeno* if $|\mathit{Untimed}(\rho)| = \omega$ and $\mathit{Duration}(\rho) = \mathit{Duration}(\mathit{trace}(\rho)) = r$ with $r \in \mathbb{R}_{\geq 0}$.
A run is *initial* if $\mathit{first}(\rho) \in Q_0$. A run ρ is *accepting* if *i)* either ρ is a finite initial run and $\mathit{last}(\rho) \in F$ or *ii)* ρ is an infinite initial run and there is a state $q \in R$ that appears infinitely often on ρ. A timed word $w = (a_i, d_i)_{i\geq 0}$ is *accepted* by S if there is an accepting run ρ s.t. trace $\mathit{trace}(\rho) = w$. The *timed language*, $\mathcal{L}(S)$, accepted by S is the set of finite and infinite timed words accepted by S. We let $\mathcal{L}^*(S)$ (resp. $\mathcal{L}^\omega(S)$) be the set of finite (resp. infinite) timed words accepted by S. When we omit the sets F and R, it means that $F = R = Q$ i.e., every state is final and repeated. In this case the language accepted by the TTS is *prefix-closed*, that is, if the TTS accepts a timed word w, then it also accepts every finite prefix of w.

2.2. Equivalences on Timed Transition Systems

We can define two types of equivalences on TTS: roughly speaking, language equivalences are based on the set of timed words two TTS generate. Branching equivalences (like simulation or bisimulation) are finer in the sense that they involve the branching structure of the two TTS.

Definition 3 (Timed Language Equivalence) *Let* $S_i = (Q_i, Q_0^i, A, \longrightarrow_i, F_i, R_i)$ *with* $i = 1, 2$ *be two TTS.* S_1 *and* S_2 *are* language equivalent*, denoted* $S_1 =_{\mathcal{L}} S_2$*, if* $\mathcal{L}(S_1) = \mathcal{L}(S_2)$.

Branching equivalences can be defined on TTS and are more constraining:

Definition 4 (Strong Timed Similarity) *Let* $S_1 = (Q_1, Q_0^1, A, \longrightarrow_1, F_1, R_1)$ *and* $S_2 = (Q_2, Q_0^2, A, \longrightarrow_2, F_2, R_2)$ *be two TTS and* $\preceq$ *be a binary relation over* $Q_1 \times Q_2$. *We write* $s \preceq s'$ *for* $(s, s') \in \preceq$. $\preceq$ *is a* strong (timed) simulation relation *of* S_1 *by* S_2 *if:*

1. *if* $s_1 \in F_1$ *(resp.* $s_1 \in R_1$*) and* $s_1 \preceq s_2$ *then* $s_2 \in F_2$ *(resp.* $s_2 \in R_2$*);*
2. *if* $s_1 \in Q_0^1$ *there is some* $s_2 \in Q_0^2$ *s.t.* $s_1 \preceq s_2$*;*
3. *if* $s_1 \xrightarrow{d}_1 s_1'$ *with* $d \in \mathbb{R}_{\geq 0}$ *and* $s_1 \preceq s_2$ *then* $s_2 \xrightarrow{d}_2 s_2'$ *for some* s_2'*, and* $s_1' \preceq s_2'$*;*
4. *if* $s_1 \xrightarrow{a}_1 s_1'$ *with* $a \in A$ *and* $s_1 \preceq s_2$ *then* $s_2 \xrightarrow{a}_2 s_2'$ *and* $s_1' \preceq s_2'$.

A TTS S_2 strongly simulates S_1 *if there is a strong (timed) simulation relation of* S_1 *by* S_2. *We write* $S_1 \preceq_{\mathcal{S}} S_2$ *in this case.*

When there is a strong simulation relation $\preceq$ of S_1 by S_2 and $\preceq^{-1}$ is also a strong simulation relation[4] of S_2 by S_1, we say that $\preceq$ is a *strong (timed) bisimulation relation* between S_1 and S_2 and use $\approx$ instead of $\preceq$. Two TTS S_1 and S_2 are *strongly (timed) bisimilar* if there exists a strong (timed) bisimulation relation between S_1 and S_2. We write $S_1 \approx_{\mathcal{S}} S_2$ in this case.

Let $S = (Q, Q_0, \Sigma_\varepsilon, \longrightarrow, F, R)$ be a TTS. We define the ε-abstract TTS $S^\varepsilon = (Q, Q_0^\varepsilon, \Sigma, \longrightarrow_\varepsilon, F, R)$ (with no ε-transitions) by:

- $q \xrightarrow{d}_\varepsilon q'$ with $d \in \mathbb{R}_{\geq 0}$ iff there is a run $\rho = q \xrightarrow{*} q'$ with $Untimed(\rho) = \varepsilon$ and $Duration(\rho) = d$,
- $q \xrightarrow{a}_\varepsilon q'$ with $a \in \Sigma$ iff there is a run $\rho = q \xrightarrow{*} q'$ s.t. $Untimed(\rho) = a$ and $Duration(\rho) = 0$,
- $Q_0^\varepsilon = \{q \mid \exists q' \in Q_0 \mid q' \xrightarrow{*} q \text{ and } Duration(\rho) = 0 \wedge Untimed(\rho) = \varepsilon\}$.

Definition 5 (Weak Timed Similarity) *Let* $S_i = (Q_i, Q_0^i, \Sigma_\varepsilon, \longrightarrow_i, F_i, R_i)$ *for* $i = 1, 2$ *be two TTS and* $\preceq$ *be a binary relation over* $Q_1 \times Q_2$. $\preceq$ *is a* weak (timed) simulation relation *of* S_1 *by* S_2 *if it is a strong timed simulation relation of* S_1^ε *by* S_2^ε. *A TTS* S_2 weakly simulates S_1 *if there is a weak (timed) simulation relation of* S_1 *by* S_2. *We write* $S_1 \preceq_{\mathcal{W}} S_2$ *in this case.*

When there is a weak simulation relation $\preceq$ of S_1 by S_2 and $\preceq^{-1}$ is also a weak simulation relation of S_2 by S_1, we say that $\preceq$ is a *weak (timed) bisimulation relation* between S_1 and S_2 and use $\approx$ instead of $\preceq$. Two TTS S_1 and S_2 are *weakly (timed) bisimilar* if there exists a weak (timed) bisimulation relation between S_1 and S_2. We write $S_1 \approx_{\mathcal{W}} S_2$ in this case. Note that if $S_1 \preceq_{\mathcal{S}} S_2$ then $S_1 \preceq_{\mathcal{W}} S_2$ and if $S_1 \preceq_{\mathcal{W}} S_2$ then $\mathcal{L}(S_1) \subseteq \mathcal{L}(S_2)$.

3. Time Petri Nets

Time Petri Nets were introduced in (Merlin, 1974) and extend Petri Nets with timing constraints on the firings of transitions. In this section, we give the definitions and semantics of an extended class of TPNs using open and/or closed intervals (Bérard et al., 2005a; Cassez and Roux, 2006).

[4] $s_2 \preceq^{-1} s_1 \iff s_1 \preceq s_2$.

3.1. Definition and Semantics

Definition 6 (Time Petri Net) *A* Time Petri Net *(TPN)* $\mathcal{T}$ *is a tuple* $(P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ *where:*

- $P = \{p_1, p_2, \cdots, p_m\}$ *is a finite set of* places*;*
- $T = \{t_1, t_2, \cdots, t_n\}$ *is a finite set of* transitions*;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the* backward *incidence mapping;* $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the* forward *incidence mapping;*
- $M_0 \in \mathbb{N}^P$ *is the* initial *marking;*
- $\alpha \in (\mathbb{Q}_{\geq 0})^T$ *and* $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^T$ *are respectively the* earliest *and* latest *firing time mappings.*

A labeled *TPN is a pair* $(\mathcal{T}, L)$ *where* $L : T \to \Sigma_\varepsilon$.

The semantics of TPNs can be given by a Timed Transition System. $\nu \in (\mathbb{R}_{\geq 0})^n$ is a *valuation* such that each value ν_i is the elapsed time since transition t_i was last enabled. $\mathbf{0}$ is the initial valuation with $\forall i \in [1..n], \mathbf{0}_i = 0$. A *marking* M of a TPN is a mapping in $\mathbb{N}^P$ and if $M \in \mathbb{N}^P$, $M(p_i)$ is the number of tokens in place p_i. A transition t_i is *enabled* in a marking M iff $M \geq {}^\bullet t_i$ and $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$. The predicate $\uparrow enabled(t_k, M, t_i) \in \mathbb{B}$ is true if t_k is enabled by the firing of transition t_i from marking M, and false otherwise. This definition of enabledness is based on (Berthomieu and Diaz, 1991; Aura and Lilius, 2000) which is the most common one. In this framework, a transition t_k is *newly enabled* after firing t_i from marking M if "it is not enabled by $M - {}^\bullet t_i$ and is enabled by $M' = M - {}^\bullet t_i + t_i{}^\bullet$" (Berthomieu and Diaz, 1991). Formally this gives:

$$\uparrow Enabled(t_k, M, t_i) = \left(M - {}^\bullet t_i + t_i{}^\bullet \geq {}^\bullet t_k\right) \wedge \left((M - {}^\bullet t_i < {}^\bullet t_k) \vee (t_k = t_i)\right) \tag{1}$$

Definition 7 (Semantics of a TPN) *The semantics of a TPN* $\mathcal{T}$ *is a timed transition system* $S_\mathcal{T} = (Q, q_0, T, \to)$ *where:* $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$, $q_0 = (M_0, \mathbf{0})$, $\longrightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ *consists of:*

- *the discrete transition relation is defined for all* $t_i \in T$ *by* $(M, \nu) \xrightarrow{t_i} (M', \nu')$ *iff:*

$$\begin{cases} M \geq {}^\bullet t_i \wedge M' = M - {}^\bullet t_i + t_i{}^\bullet \\ \alpha(t_i) \leq \nu_i \leq \beta(t_i) \\ \nu'_k = \begin{cases} 0 & \text{if } \uparrow Enabled(t_k, M, t_i), \\ \nu_k & \text{otherwise.} \end{cases} \end{cases}$$

- *and the continuous transition relation is defined for all* $d \in \mathbb{R}_{\geq 0}$ *by* $(M, \nu) \xrightarrow{d} (M, \nu')$ *iff:*

$$\begin{cases} \nu' = \nu + d \\ \forall k \in [1..n], \left(M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k)\right) \end{cases}$$

A *run* of a time Petri net $\mathcal{T}$ is a (finite or infinite) path in $S_\mathcal{T}$ starting in q_0. The set of runs of $\mathcal{T}$ is denoted by $Runs(\mathcal{T})$. As a shorthand we write $(M, \nu) \longrightarrow^d_e (M', \nu')$ for a sequence of time elapsing and discrete steps like $(M, \nu) \xrightarrow{d} (M'', \nu'') \xrightarrow{e} (M', \nu')$. A state (M, ν) is reachable in $\mathcal{T}$ if $(M_0, \nu_0) \xrightarrow{*} (M, \nu)$. $ReachState(\mathcal{T})$ is the reachable set of states in $\mathcal{T}$. A marking M is

reachable in $\mathcal{T}$ if there is a state $(M, \nu) \in \mathit{ReachState}(\mathcal{T})$. The set of *reachable markings* of $\mathcal{T}$ is denoted $\mathit{ReachMark}(\mathcal{T})$. If the set $\mathit{ReachMark}(\mathcal{T})$ is finite we say that $\mathcal{T}$ is *bounded*, otherwise it is *unbounded*.

If we add two sets of markings, F (final) and R (repeated), we can define the languages accepted by $\mathcal{T}$. We let S_F (resp. S_R) be the set of states (M, ν) of $S_{\mathcal{T}}$ s.t. $M \in F$ (resp. $M \in R$). The timed language $\mathcal{L}(\mathcal{T})$ accepted by $\mathcal{T}$ is the timed language accepted by $S_{\mathcal{T}}$ with sets S_F and S_R as final and repeated states. Similar definitions hold for $\mathcal{L}^*(\mathcal{T})$ and $\mathcal{L}^\omega(\mathcal{T})$. Moreover, for a labeled TPN $(\mathcal{T}, L)$, the languages accepted by $(\mathcal{T}, L)$ are the languages accepted by $\mathcal{T}$ where, in each timed word, a transition label t is replaced by $L(t)$.

In Definition 7, we have implicitly assumed that the constraints given on each transition of the TPN by the mapping (α, β) are *closed constraints*. We can also consider that the firing intervals of a transition are left and/or right open. The semantics is defined accordingly substituting $<$ to $\leq$: assume $\alpha(t_i), \beta(t_i)$ is left-closed and right open; in this case the discrete transition relation for t_i is defined using $\alpha(t_i) \leq \nu_i < \beta(t_i)$ instead of $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$ in Definition 7 and for the continuous transition relation, we should use $\nu_i' < \beta(t_i)$. Most of the results we give in this chapter hold for any type of intervals. In the sequel, we denote $\mathcal{TPN}$ the most general class of TPNs using open or closed intervals and $\mathcal{TPN}(\leq, \geq)$ for the subclass that uses only closed intervals.

Our semantics (Bérard et al., 2005a; Cassez and Roux, 2006) is based on the common definition of (Berthomieu and Diaz, 1991; Aura and Lilius, 2000) for safe TPNs but still there are some advantages using ours. First, previous formal semantics (Berthomieu and Diaz, 1991; Lilius, 1998; Pezzè, 1999; Aura and Lilius, 2000) for TPNs required the TPNs to be *safe*. Our semantics encompasses the whole class of TPNs and is fully consistent with the previous semantics when restricted to safe TPNs[5]. Thus, we have given a semantics to multiple enabledness of transitions which seems the most simple and adequate. Indeed, several interpretations can be given to multiple enabledness (Berthomieu and Diaz, 1991).

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. The paper (Pezzè, 1999) considers two distinct semantics: Weak Time Semantics (WTS) and Strong Time Semantics (STS). According to WTS, a transition *can* be fired only in its time interval whereas in STS, a transition *must* fire within its firing interval unless disabled by the firing of others. The most commonly used semantics is STS as in (Merlin, 1974; Berthomieu and Diaz, 1991; Pezzè, 1999; Aura and Lilius, 2000). A more complete study on the firing policy in TPNs can be found in (Bérard et al., 2005c).

Third, it is possible for the TPN to generate zeno runs or to be unbounded. When it is unbounded, the discrete component (i.e., marking) of the state space of the timed transition system is infinite. If $\forall i, \alpha(t_i) > 0$ then the TPN cannot generate any zeno word and time diverges on each run. Otherwise, if the TPN is bounded and at least one lower bound is 0, whether or not a TPN accepts a zeno run can be decided (Henzinger et al., 1994) (for instance using the equivalent timed automaton we build in section 4.3). In the next subsections we summarize the status of basic decision problems for TPNs.

3.2. Decidable and Undecidable Problems for TPNs

Let $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ be a TPN with $|P| = p$, $|T| = n$ and let $S_{\mathcal{T}} = (Q, q_0, T, \to)$ its semantics. Let us consider the following problems:

(1) The *marking reachability* problem: Given $m \in \mathbb{N}^p$, is m in $\mathit{ReachMark}(\mathcal{T})$?

(2) The *boundedness* problem: Is there a bound $\overline{\boldsymbol{b}} \in \mathbb{N}^p$ s.t. $\forall m \in \mathit{ReachMark}(\mathcal{T})$, $m \leq \overline{\boldsymbol{b}}$?

[5] If we except the difference with (Lilius, 1998) in the definition of the reset instants for newly enabled transitions.

(3) The $\overline{k}$-*boundedness* problem: Given $\overline{k} = (k_1, k_2, \cdots, k_p) \in \mathbb{N}^p$, is it true that for all $m \in ReachMark(\mathcal{T})$, $m \leq \overline{k}$?

(4) The *state reachability* problem: Given $(M, \nu) \in \mathbb{N}^P \times \mathbb{R}^n_{\geq 0}$, is (M, ν) in $ReachState(\mathcal{T})$?

(5) The *liveness* problem: For $t \in T$, $(M, \nu) \in ReachState(\mathcal{T})$, is there any run $(M, \nu) \xrightarrow{*} (M', \nu')$ such that $(M', \nu') \xrightarrow{t} (M'', \nu'')$?

(6) The *emptiness* problem: Is the language accepted by $\mathcal{T}$ empty i.e., $\mathcal{L}(S_{\mathcal{T}}) = \emptyset$?

(7) The *universal* problem: Does $\mathcal{T}$ accept all the finite (resp. infinite) timed words over T i.e., $\mathcal{L}^*(\mathcal{T}) = \mathrm{TW}^*(T)$ (resp. $\mathcal{L}^\omega(\mathcal{T}) = \mathrm{TW}^\omega(T)$)?

Problem (1) was proved undecidable for TPNs in (Jones et al., 1977). It follows that all problems (1–2) and (4–5) are undecidable for TPNs. There are however a number of sufficient conditions for the boundedness property of TPNs, the stronger being that the underlying Petri net is bounded, and the latter is known decidable. For Time Petri nets, we have the following results:

Theorem 1 (Berthomieu and Diaz, 1991) $\overline{k}$*-boundedness* (3) *is decidable for TPNs. State reachability* (4) *and liveness* (5) *are decidable for bounded TPNs.*

Theorem 2 (Cassez and Roux, 2006) *The emptiness problem* (6) *is PSPACE-complete for bounded TPNs.*

Theorem 3 (Bérard et al., 2005a) *The universal problem* (7) *is undecidable for bounded TPNs.*

Theorem 2 is obtained by reducing the emptiness problem for bounded TPNs to the emptiness problem for TA (using the result of Section 4). Theorem 3 is more difficult to obtain and consists in translating a TA into an equivalent TPN (using the results of Table 1). In the next two subsections we describe two ways of solving the *state reachability (4)* problem.

3.3. State Reachability Using The State Class Method

To decide problem 4, we can compute a finite representation of the state space of a bounded TPN. If, on this representation G, we can decide whether a state (M, ν) belongs to G we have an algorithm to check state reachability.
The first method to compute the state space of a TPN is based on the aggregation of states into classes and was introduced by BERTHOMIEU and DIAZ in (Berthomieu and Diaz, 1991).

Definition 8 (State Class) *A State Class C of a TPN is a pair (M, D) where M is a marking and D is a set of inequalities, over a set of variables X, called the firing domain. The value of a variable $x_i \in X$ of the firing domain represents the firing time of the enabled transition t_i relatively to the time when the class C was entered.*

To obtain an abstract representation of the state space of a TPN, it is possible to compute a graph of state classes called the *State Class Graph (SCG)*. An edge in the SCG from a class to another is defined by:

Definition 9 (State Class Transition) *Given a class $C = (M, D)$ and a transition t_j enabled in (M, D), the t_j-successor class, $C' = (M', D')$, of C is computed as follows:*

1. *Compute the new marking $M' = M - {}^\bullet t_j + t_j^\bullet$;*

2. *Add the constraints $x_j \leq x_i$ for each $i \neq j$ to D. Then substitute in D each variable $x_i, i \neq j$ by $x'_i + x_j$ where x'_i are new fresh variables. The new domain obtained this way is D';*
3. *Eliminate x_j from D' using for instance the Fourier-Motzkin method;*
4. *Replace x'_i by x_i in D'.*
5. *Compute a canonical form of D' using for instance the Floyd-Warshall algorithm.*

Computing all the edges of the SCG from the initial class of a TPN is called the *State-Class Method*. In the state class method, the domain associated with a class is relative to the time when the class was entered and as the transformation (we reset the time origin) is irreversible, absolute values of clocks cannot be obtained easily. The graph produced is an abstraction of the state space for which temporal information has been lost. Often, the graph has more classes than the number of markings of the TPN. Edges between classes are no longer labeled with a firing constraint but only with the name of the fired transition: the state class graph accepts the untimed (prefix closed) language of the TPN. If the TPN is bounded the SCG of a TPN is finite. The SCG computation is implemented in the tool TINA (Berthomieu and Vernadat, 2006a).
As a consequence of the SCG construction, sophisticated temporal properties are not easy to check. Indeed, the domain associated with a marking is made of relative values of clocks and the function to compute domains is not bijective. Consequently, domains cannot be easily used to verify properties involving constraints on clocks.
In order to get rid of these limitations, several papers have proposed to construct a different state class graph by modifying the equivalence relation between classes. To our knowledge, the methods proposed in (Berthomieu and Vernadat, 2003) depend on the property to check. Checking LTL or CTL properties will lead to different state class graphs.
Another limitation of the methods and associated tools to check properties of TPN using the SCG, is the need to compute the whole state graph while only the reachability of a given marking is needed (safety properties). The graph is then analyzed by a model checker for finite state systems. Using observers is even more costly: actually, for each property to be checked, a new state class graph has to be built and the observer can dramatically increase the size of the state space.

3.4. State Reachability Using a Zone Based Abstraction

Another method to compute a finite representation of the state space of a bounded TPN was recently proposed by GARDEY *et al.* in (Gardey et al., 2003; Gardey et al., 2006). It is based on the Region Graph introduced for Timed Automata (Alur and Dill, 1994; Rokicki, 1993).
A *zone* is a convex union of regions as defined by ALUR and DILL (Alur and Dill, 1994). For short, considering n clocks, a zone is a convex subset of $\mathbb{R}^n_{\geq 0}$. A zone can be represented by a conjunction of constraints on pairs of clocks: $x_i - x_j \sim c$ where $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.
The graph which is computed in this case is a *simulation graph* of a TPN which is an abstract and *symbolic* representation of the state space of the TPN. Given the initial marking M_0 and an initial zone Z_0 (the values of clocks for Z_0 are $\mathbf{0}$), time and discrete successors are iteratively computed by letting time pass or by firing transitions. Let M be a marking and Z a zone. The computation of the reachable markings from (M, Z) is done as follows:

1. Compute the possible states reachable by time elapsing: we let $\overrightarrow{Z}$ be the set of such states. It is obtained by setting all upper bounds of constraints on clocks defining Z to infinity;

2. Select only the possible valuations of clocks for which M could exist, i.e., the valuations of clocks are smaller than the latest firing time of any enabled transitions;

$$Z' = \overrightarrow{Z} \cap \bigwedge_{t_i \in Enabled M,Z} \{x_i \leq \beta_i\}$$

where $\{x \leq \beta\}$ denotes the zone defined by the constraint $x \leq \beta$. Z' is the maximal zone starting from Z for which the marking M exists.

3. Determine the firable transitions in (M, Z'): t_i is firable if $Z' \cap \{x_i \geq \alpha_i\}$ is a non empty zone.

4. For each firable transition t_i leading to a marking M_i, compute the zone obtained when we enter the new marking M_i as follows:

$$Z_i = (Z' \cap \{x_i \geq \alpha_i\})[X_e := 0]$$

where X_e is the set of newly enabled clocks. This means that each transition which is newly enabled has its clock reset. Then, Z_i is a zone for which the new marking M_i is reachable.

It is then possible to compute all the reachable pairs (M, Z) reachable from (M_0, Z_0) using the previous method. This way we obtain a forward algorithm to compute the simulation graph for a bounded TPN.
An algorithm to enumerate reachable markings for a bounded TPN could be based on the previous iterative process but, in some cases, it will lead to a non-terminating computation. Though the number of reachable markings is finite for a bounded TPN, the number of zones in which a marking is reachable is not necessarily finite as shown by the TPN $\mathcal{T}_0$ in Fig. 1.

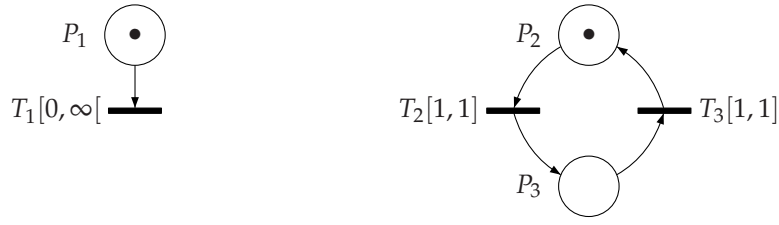


Figure 1. A TPN $\mathcal{T}_0$ with an Unbounded Number of Zones

The initial zone of $\mathcal{T}_0$ is Z_0 is $\{x_1 = 0 \wedge x_2 = 0 \wedge x_3 = 0\}$ (where x_i is the clock associated to T_i) and the initial marking $M_0 = (P_1, P_2, P_3) = (1, 1, 0)$. Consider the infinite firing sequence: $(T_2.T_3)^\omega$. By letting time pass, M_0 is reachable until $x_2 = 1$. When $x_2 = x_1 = 1$ the transition T_2 has to be fired. The zone corresponding to these clock values is: $Z_0 = \{0 \leq x_1 \leq 1 \wedge x_1 - x_2 = 0\}$. By firing T_2 and then T_3, $\mathcal{T}_0$ reaches its initial marking M_0. When it enters M_0, the values of (new) clocks are: $x_1 = 2$, $x_2 = 0$ and $x_1 - x_2 = 2$. Indeed, T_1 remains enabled while T_2 and T_3 are fired and x_2 is reset when T_3 is fired because T_2 became newly enabled. Given these new values, the initial marking can exist while $x_2 \leq 1$ i.e., for the zone: $Z_1 = \{2 \leq x_1 \leq 3 \wedge x_1 - x_2 = 2\}$. By applying infinitely the sequence $T_2.T_3$, there exists an infinite number of zones for which the initial marking is reachable.

Actually, the number of zones is not bounded because infinity is used as latest firing time for T_1. If for all the transitions t_i of a TPN, $\beta(t_i) \in \mathbb{Q}_{\geq 0}$, i.e., the upper bound is finite, we say that the TPN is *t-bounded*. if a TPN is t-bounded, all the clocks in the simulation graph are bounded and so, the number of different zones is bounded (Alur and Dill, 1994). The algorithm computing the simulation graph terminates in this case and it gives a finite (exact) representation of the state space of a bounded TPN.
We now present a more general algorithm which computes the state space of a TPN as defined in section 2, i.e., even if the TPN is not t-bounded. It is based on the use of an operator on zones which constructs equivalence classes. The resulting equivalence relation will be of finite index. A common operator on zones is the *k-approx* operator. For a given integer k, the use of this operator allows to create a finite set of distinct zones as presented in (Alur and Dill, 1994). To compute the simulation graph, we refine step 4 of the previous computation algorithm by applying the *k-approx* operator on the zone resulting from this last step.
This approximation is based on the fact that once the clock associated with an "unbounded" transition ($[\alpha, \infty[$) has reached the value α, its precise value does not matter. Using *k-approx* (with $k = \alpha$) allows to group all zones $[x, \infty[, x \geq \alpha$ in one equivalence class.
Previous papers on Timed Automata (Bouyer, 2004; Bouyer, 2003) have proved that this operator generally leads to a strict upper-approximation of the reachable state space. Nevertheless, for a given class of TA called *diagonal-free* TA, there is no upper-approximation of the reachable markings (Bouyer, 2004; Bouyer, 2003), and this also holds for TPNs:

Theorem 4 *For a bounded TPN, the (forward) algorithm to compute the simulation graph using k-approx on zones is exact (with respect to marking reachability) and terminates.*

In other words, checking whether $M \in ReachMark(\mathcal{T})$ is equivalent to checking whether there is a state class $C = (M, D)$ in the simulation graph. As the approximation is only needed for TPNs where some transitions have infinity as latest firing time, the following corollary holds:

Corollary 1 *For a bounded and t-bounded TPN, the (forward) algorithm to compute the simulation graph using zones is exact (with respect to marking reachability) and terminates.*

4. Comparison of Time Petri Nets and Timed Automata

In this section we give some results concerning the expressive power of TPNs and *Timed automata* (Alur and Dill, 1994). Timed automata (TA) are very similar to TPNs and a lot of theoretical results have been obtained for TA. Moreover efficient tools have been developed to check real-time properties on this model. It is thus important to compare the two formalisms and see if they can provide new insights for TPNs.

4.1. Timed Automata and Products of Timed Automata

Timed automata were studied by ALUR and DILL (Alur and Dill, 1994) and are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 10 (Timed Automaton) *A* Timed Automaton *H is a tuple (N, l_0, C, A, E, Inv) where:*

- *N is a finite set of* locations*;*
- *$l_0 \in N$ is the* initial location*;*
- *X is a finite set of positive real-valued* clocks*;*
- *A is a finite set of* actions*;*

- $E \subseteq N \times \mathcal{C}(C) \times A \times 2^X \times N$ *is a finite set of* edges, $e = \langle l, \gamma, a, R, l' \rangle \in E$ *represents an edge from the location* l *to the location* l' *with the guard* γ*, the label* a *and the reset set* $R \subseteq X$*;*
- $Inv \in \mathcal{C}(X)^N$ *assigns an* invariant *to any location. We restrict the invariants to conjuncts of terms of the form* $c \leq r$ *for* $c \in C$ *and* $r \in \mathbb{N}$*.*

The semantics of a timed automaton is a timed transition system.

Definition 11 (Semantics of a Timed Automaton) *The semantics of a timed automaton* $H = (N, l_0, X, A, E, Inv)$ *is given by a timed transition system* $S_H = (Q, q_0, \rightarrow)$ *with* $Q = N \times \mathbb{R}^X_{\leq 0}$*,* $q_0 = (l_0, \mathbf{0})$ *is the initial state and* $\rightarrow$ *consists of the discrete and continuous transition relations:*

- *the discrete transition relation if defined for all* $a \in A$ *by* $(l, v) \xrightarrow{a} (l', v')$ *if:*

$$\exists\, (l, \gamma, a, R, l') \in E \text{ s.t. } \begin{cases} \gamma(v) = \mathit{tt}, \\ v' = v[R \mapsto 0] \\ Inv(l')(v') = \mathit{tt} \end{cases}$$

- *the continuous transitions is defined for all* $t \in \mathbb{R}_{\geq 0}$ *by* $(l, v) \xrightarrow{t} (l', v')$ *if:*

$$\begin{cases} l = l' \qquad v' = v + t \quad \textit{and} \\ \forall\, 0 \leq t' \leq t,\ Inv(l)(v + t') = \mathit{tt} \end{cases}$$

A run *of a timed automaton* H *is an initial run in* S_H *starting in* q_0*. The set of runs of* H *is denoted by* $Runs(H)$*. If we add two sets of locations* $F \subseteq N$ *and* $R \subseteq N$ *we can define the timed languages accepted by a TA* H*. We let* $\mathcal{L}(H)$*,* $\mathcal{L}^*(H)$ *and* $\mathcal{L}^\omega(H)$ *be the different timed languages accepted by* H*.*

Modularity is important for modeling systems and it is convenient to describe a system as a parallel composition of timed automata. To this end, we use the classical composition notion based on a *synchronization function* à la Arnold-Nivat. Let $X = \{x_1, \cdots, x_n\}$ be a set of clocks, $H_1, \ldots, H_n$ be n timed automata with $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$. A *synchronization function* f is a partial function from $(A \cup \{\bullet\})^n \hookrightarrow A$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(H_1 | \ldots | H_n)_f$ the parallel composition of the H_i's w.r.t. f. The configurations of $(H_1 | \ldots | H_n)_f$ are pairs $(\mathbf{l}, \mathbf{v})$ with $\mathbf{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $\mathbf{v} = (v_1, \cdots, v_n)$ where each v_i is the value of the clock $x_i \in X$. Then the semantics of a synchronized product of timed automata is also a timed transition system: the synchronized product can do a discrete transition if all the components agree to do so, and time can progress in the synchronized product also if all the components agree to do so. This is formalized by the following definition:

Definition 12 (Semantics of a Product of Timed Automata) *Let* $H_1, \ldots, H_n$ *be timed automata with* $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$*, and* f *a (partial) synchronization function* $(A \cup \{\bullet\})^n \hookrightarrow A$*. The semantics of* $(H_1 | \ldots | H_n)_f$ *is a timed transition system* $S = (Q, q_0, A, \rightarrow)$ *with* $Q = N_1 \times \ldots \times N_n \times \mathbb{R}^X_{\geq 0}$*,* q_0 *is the initial state* $((l_{1,0}, \ldots, l_{n,0}), \mathbf{0})$ *and* $\rightarrow$ *is defined by:*

- $(\mathbf{l}, \mathbf{v}) \xrightarrow{b} (\mathbf{l}', \mathbf{v}')$ *if there exists* $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ *s.t.* $f(a_1, \ldots, a_n) = b$ *and for any* i *we have:*

. *If* $a_i = \bullet$, *then* $\mathbf{l}'[i] = \mathbf{l}[i]$ *and* $\mathbf{v}'[i] = \mathbf{v}[i]$,

. *If* $a_i \in A$, *then* $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{a_i} (\mathbf{l}'[i], \mathbf{v}'[i])$.

- $(\mathbf{l}, \mathbf{v}) \xrightarrow{t} (\mathbf{l}, \mathbf{v}')$ *if for all* $i \in [1..n]$, *every* H_i *agrees on time elapsing i.e.,* $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{t} (\mathbf{l}[i], \mathbf{v}'[i])$.

We could equivalently define the product of n timed automata syntactically, building a new timed automaton from the n initial ones. In the sequel, we consider a product $(H_1|\ldots|H_n)_f$ to be a timed automaton the semantics of which is timed bisimilar to the semantics of the product we have given in Definition 12.

4.2. Expressiveness of TA vs TPNs

In this subsection, we define some criteria to compare the expressive power of TA and TPNs. We then show how to translate a TPN into an equivalent TA.

4.2.1. Expressiveness and Equivalence Problems

If B, B' are either TPNs or TA, we write $B \approx_{\mathcal{S}} B'$ (resp. $B \approx_{\mathcal{W}} B'$) for $S_B \approx_{\mathcal{S}} S_{B'}$ (resp. $S_B \approx_{\mathcal{W}} S_{B'}$). Let $\mathcal{C}$ and $\mathcal{C}'$ be two classes of TPNs or TA.

Definition 13 (Expressiveness w.r.t. Timed Language Acceptance) *The class* $\mathcal{C}$ *is* more expressive *than* $\mathcal{C}'$ *w.r.t. timed language acceptance if for all* $B' \in \mathcal{C}'$ *there is a* $B \in \mathcal{C}$ *s.t.* $\mathcal{L}(B) = \mathcal{L}(B')$. *We write* $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$ *in this case. If moreover there is some* $B \in \mathcal{C}$ *s.t. there is no* $B' \in \mathcal{C}'$ *with* $\mathcal{L}(B) = \mathcal{L}(B')$, *then* $\mathcal{C}' <_{\mathcal{L}} \mathcal{C}$ *(read "strictly more expressive"). If both* $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$ *and* $\mathcal{C} \leq_{\mathcal{L}} \mathcal{C}'$ *then* $\mathcal{C}$ *and* $\mathcal{C}'$ *are equally expressive w.r.t. timed language acceptance, and we write* $\mathcal{C} =_{\mathcal{L}} \mathcal{C}'$.

Definition 14 (Expressiveness w.r.t. Timed Bisimilarity) *The class* $\mathcal{C}$ *is* more expressive *than* $\mathcal{C}'$ *w.r.t. strong (resp. weak) timed bisimilarity if for all* $B' \in \mathcal{C}'$ *there is a* $B \in \mathcal{C}$ *s.t.* $B \approx_{\mathcal{S}} B'$ *(resp.* $B \approx_{\mathcal{W}} B'$*). We write* $\mathcal{C}' \leq_{\mathcal{S}} \mathcal{C}$ *(resp.* $\mathcal{C}' \leq_{\mathcal{W}} \mathcal{C}$*) in this case. If moreover there is a* $B \in \mathcal{C}$ *s.t. there is no* $B' \in \mathcal{C}'$ *with* $B \approx_{\mathcal{S}} B'$ *(resp.* $B \approx_{\mathcal{W}} B'$*), then* $\mathcal{C}' <_{\mathcal{S}} \mathcal{C}$ *(resp.* $\mathcal{C}' <_{\mathcal{W}} \mathcal{C}$*). If both* $\mathcal{C}' <_{\mathcal{S}} \mathcal{C}$ *and* $\mathcal{C} <_{\mathcal{S}} \mathcal{C}'$ *(resp.* $<_{\mathcal{W}}$*) then* $\mathcal{C}$ *and* $\mathcal{C}'$ *are equally expressive w.r.t. strong (resp. weak) timed bisimilarity, and we write* $\mathcal{C} \approx_{\mathcal{S}} \mathcal{C}'$ *(resp.* $\mathcal{C} \approx_{\mathcal{W}} \mathcal{C}'$*).*

In the sequel we will compare various classes of TPNs and TAs. When refering to language acceptance we assume that two sets F and R have been given for a TPN (see Definition 7) and for a TA. We use the following notations:

- B-$\mathcal{TPN}_\varepsilon$ for the set of bounded labeled TPNs with ε-transitions (Definition 7);
- 1-B-$\mathcal{TPN}_\varepsilon$ for the subset of B-$\mathcal{TPN}_\varepsilon$ with at most one token in each place (one safe TPN);
- B-$\mathcal{TPN}(\leq, \geq)$ for the subset of B-$\mathcal{TPN}_\varepsilon$ where only closed intervals are used;
- $\mathcal{TA}_\varepsilon$ for TA with ε-transitions; $\mathcal{TA}^*_{(\leq,\geq)}$ for the syntactical subclass of TA that is equivalent to B-$\mathcal{TPN}(\leq, \geq)$ (see (Bérard et al., 2005a)).

$\mathcal{TA}^*_{(\leq,\geq)}$ is formally defined by:

Definition 15 *The subclass* $\mathcal{TA}^*_{(\leq,\geq)}$ *of TA is defined by the set of TA of the form* (L, l_0, X, A, E, Inv) *where :*

	Timed Language Acceptance	Timed Bisimilarity
B-$\mathcal{TPN}_\varepsilon$	$\leq_{\mathcal{L}} \mathcal{TA}_\varepsilon$ (Cassez and Roux, 2006)	$\leq_{\mathcal{W}} \mathcal{TA}_\varepsilon$ (Cassez and Roux, 2006)
	$=_{\mathcal{L}} \mathcal{TA}_\varepsilon$ (Bérard et al., 2005a)	$<_{\mathcal{W}} \mathcal{TA}_\varepsilon$ (Bérard et al., 2005a)
B-$\mathcal{TPN}(\leq,\geq)$	$=_{\mathcal{L}} \mathcal{TA}^*_{(\leq,\geq)}$ (Bérard et al., 2005b)	$\approx_{\mathcal{W}}$ 1-B-$\mathcal{TPN}(\leq,\geq)$ (Bérard et al., 2005b) $\approx_{\mathcal{W}} \mathcal{TA}^*_{(\leq,\geq)}$ (Bérard et al., 2005b)

Table 1. Summary of the Expressiveness Results for TPNs vs. TA

- *guards are conjunctions of atomic constraints of the form $x \geq c$ and invariants are conjunction of atomic constraints $x \leq c$.*
- *the invariants satisfy the following property; $\forall e = (\ell, \gamma, a, R, \ell') \in E$, if $x \notin R$ and $x \leq c$ is an atomic constraint in $Inv(\ell)$, then if $x \leq c'$ is $Inv(\ell')$ for some c' then $c' \geq c$.*

In Table 1, $\preceq_{\mathcal{L}}$ or $\preceq_{\mathcal{W}}$ with $\preceq \in \{<, \leq\}$, respectively means "less expressive than" w.r.t. Timed Language Acceptance and Weak Timed Bisimilariy; the term $=_{\mathcal{L}}$ means "equally expressive as" w.r.t. language acceptance and $\approx_{\mathcal{W}}$ "equally expressive as" w.r.t. weak timed bisimilarity. A consequence of the results in this table is that 1-B-$\mathcal{TPN}_\varepsilon$ and B-$\mathcal{TPN}_\varepsilon$ are equally expressive w.r.t. Timed Language Acceptance i.e., 1-B-$\mathcal{TPN}_\varepsilon$ $=_{\mathcal{L}}$ B-$\mathcal{TPN}_\varepsilon$. An equivalent result was known for untimed PN (we can always obtain a safe PN that accepts the same language as a PN) but the counterpart for TPN was proved in (Bérard et al., 2005a).
Surprisingly, bounded TPNs are less expressive than timed automata w.r.t. timed bisimulation. We will see in subsection 4.3 how to translate a TPN into a timed bisimilar TA. Thanks to this translation, we will use in section 5 the TA obtained to check TCTL properties of the original TPN.

4.3. From Time Petri Nets to Timed Automata

The relationship between TPNs and TA has not been much investigated before 2000. In (Sifakis and Yovine, 1996) J. SIFAKIS and S. YOVINE are mainly concerned with *compositionality* problems. They show that for a subclass of 1-safe Time Stream Petri Nets, the usual notion of composition used for TA is not suitable to describe this type of Petri Nets as the composition of TA. Consequently, they propose Timed Automata with Deadlines and flexible notions of composition. In (Bornot et al., 1998) the authors consider Petri nets with deadlines (PND) that are 1-safe Petri nets extended with clocks. A PND is a timed automaton with deadlines (TAD) where the discrete transition structure is the corresponding marking graph. The transitions of the marking graph are subject to the same timing constraints as the transitions of the PND. The PND and the TAD have the same number of clocks. They propose a translation of safe TPN into PND with a clock for each input arc of the initial TPN. It defines (by transitivity) a translation of safe TPN into TAD (that can be considered as standard timed automata). In (Cortès et al., 2000) the authors consider an extension of Time Petri Nets (*PRES+*) and propose a translation into hybrid automata. Correctness of the translation is not proved. Moreover the method is defined only for 1-safe nets.
In another line of work, SAVA (Sava, 2001) considers bounded TPNs where the underlying Petri net is not necessarily safe and proposes an algorithm to translate the TPN into a timed automaton (one clock is needed for each transition of the original TPN). However, the author does not give any proof that this translation is correct (i.e., it preserves some equivalence relation between the semantics of the original TPN and the computed TA) and neither that the algorithm terminates (even if the TPN is bounded).

LIME and ROUX proposed an extension in (Lime and Roux, 2006) of the state class graph construction that allows to build the state class graph of a bounded TPN as a timed automaton. They prove that this timed automaton and the TPN are timed bisimilar and they also prove a relative minimality result of the number of clocks needed in the obtained automaton.
The first two approaches are structural but are limited to Petri nets whose underlying net is 1-safe. The last two approaches rely on the computation of the state space of the TPN and are limited to bounded TPNs. In this section, we consider a structural translation from TPN (not necessary bounded) to TA proposed in (Cassez and Roux, 2006). This extends the previous results in the following directions: first, we can easily prove that our translation is correct and terminates as it is a syntactic translation and it produces a timed automaton that is timed bisimilar to the TPN we started with. Notice that the timed automaton contains integer variables that correspond to the marking of the Petri net and that it may have an unbounded number of locations. However timed bisimilarity holds even in the unbounded case. In case the Petri net is bounded, we obtain a timed automaton with a finite number of locations and we can check for TCTL properties of the original TPN. Second, as it is a structural translation it does not need expensive computation (like the State Class Graph) to obtain a timed automaton. This has a practical application as it enables one to use efficient existing tools for TA to analyze TPNs.

4.3.1. Translating Time Petri Nets into Timed Automata

In this subsection, we build a synchronized product of timed automata from a TPN so that the behaviors of the two are in a one-to-one correspondence.
We start with a TPN $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ with set of places $P = \{p_1, \cdots, p_m\}$ and set of transitions $T = \{t_1, \cdots, t_n\}$.

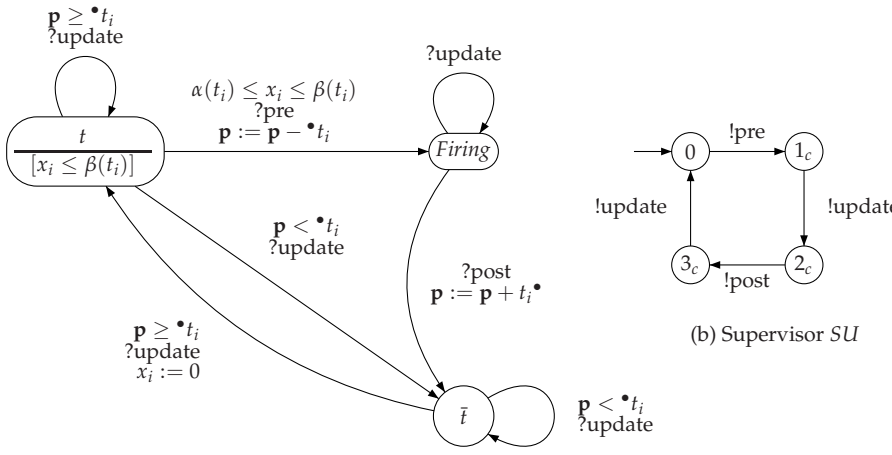
Timed Automaton Associated with a Transition

We define one timed automaton $\mathcal{A}_i$ for each transition t_i of T (see Fig. 2.a). This timed automaton has one clock x_i. Also the locations of the automaton $\mathcal{A}_i$ give the state of the transition t_i: in location t the transition is enabled; in location $\bar{t}$ it is disabled and in *Firing* it is being fired. The initial location of each $\mathcal{A}_i$ depends on the initial marking M_0 of the Petri net we want to translate. If $M_0 \geq {}^\bullet t_i$, then the initial location is t otherwise it is $\bar{t}$. This automaton updates an array of integers $\mathbf{p}$ (s.t. $\mathbf{p}[i]$ is the number of tokens in place p_i) shared by all the $\mathcal{A}_i$'s. This is not covered by Definition 12,but this extended model with integer arrays is very common (Pettersson and Larsen, 2000) and it does not affect the expressiveness of the model when the variables are bounded.

The Supervisor

The automaton for the supervisor *SU* is depicted on Fig. 2.b. The locations 1 to 3 subscripted with a "*c*" are assumed to be committed. *Committed* locations can be simulated by adding an extra variable: see (Tripakis, 1999) Appendix A for details. This means that no time can elapse while visiting them. We denote by $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n)_f$ the timed automaton associated to the TPN $\mathcal{T}$. The initial location of the supervisor is 0. Let us define the synchronization function f with $n+1$ parameters (the first element of the vector refers to the supervisor move) by:

- $f(!pre, \bullet, \cdots, ?pre, \bullet, \cdots) = pre_i$ if $?pre$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
- $f(!post, \bullet, \cdots, ?post, \bullet, \cdots) = post_i$ if $?post$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
- $f(!update, ?update, \cdots, ?update) = update$.



(a) The automaton $\mathcal{A}_i$ for transition t_i

(b) Supervisor SU

Figure 2. Automata for the Transitions and the Supervisor

In the sequel, $((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ is such that $(s, \mathbf{p}) \in \{0, 1_c, 2_c, 3_c\} \times \mathbb{N}^m$ is the state of SU, $\mathbf{q}$ gives the product location of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, and $\mathbf{v}[i], i \in [1..n]$ gives the value of the clock x_i.
We will prove in the next subsection that the semantics of $\Delta(\mathcal{T})$ is closely related to the semantics of $\mathcal{T}$. For this we have to relate the states of $\mathcal{T}$ to the states of $\Delta(\mathcal{T})$ and we define the following equivalence:

Definition 16 (State Equivalence) *Let (M, v) and $((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ be, respectively, a state of $S_\mathcal{T}$ and a configuration. Then $(M, v) \approx ((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ if:*

$$\begin{cases} s = 0, \\ \forall i \in [1..m], \quad \mathbf{p}[i] = M(p_i), \\ \forall k \in [1..n], \quad \mathbf{q}[k] = \begin{cases} t \text{ if } M \geq {}^\bullet t_k, \\ \bar{t} \text{ otherwise} \end{cases} \\ \forall k \in [1..n], \quad \mathbf{v}[k] = v_k. \end{cases}$$

4.3.2. Soundness of the Translation

We now prove that our translation preserves the behaviors of the initial TPN in the sense that the semantics of the TPN and its translation are timed bisimilar. Let $\mathcal{T}$ be a TPN and $S_\mathcal{T} = (Q, q_0, T, \rightarrow)$ its semantics. Let $\mathcal{A}_i$ be the automaton associated with transition t_i of $\mathcal{T}$ as described by Fig. 2.a, SU the supervisor automaton of Fig. 2.b and f the synchronization function defined previously. The semantics of $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n)_f$ is the TTS $S_{\Delta(\mathcal{T})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{T})}, A(\Delta(\mathcal{T})), \rightarrow)$.

Theorem 5 (Timed Bisimilarity) *For* $(M,\nu) \in S_{\mathcal{T}}$ *and* $((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \in S_{\Delta(\mathcal{T})}$ *such that* $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$ *the following holds:*

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \quad \textit{iff} \quad \begin{cases} ((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \stackrel{w_i}{\Longrightarrow} ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \textit{ with} \\ w_i = \textit{pre}_i.\textit{update}.\textit{post}_i.\textit{update and} \\ (M',\nu') \approx ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \end{cases} \tag{2}$$

$$(M,\nu) \xrightarrow{d} (M',\nu') \quad \textit{iff} \quad \begin{cases} ((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \xrightarrow{d} ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \textit{ and} \\ (M',\nu') \approx ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \end{cases} \tag{3}$$

Proof. We first prove statement (2). Assume $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$. Then as t_i can be fired from (M,ν) we have: (i) $M \geq {}^\bullet t_i$, (ii) $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$, (iii) $M' = M - {}^\bullet t_i + t_i^\bullet$, and (iv) $\nu'_k = 0$ *if* $\uparrow enabled(t_k, M, t_i)$ and $\nu'_k = \nu_k$ otherwise. From (i) and (ii) and the state equivalence we deduce that $\mathbf{q}[i] = t$ and $\alpha(t_i) \leq \mathbf{v}[i] \leq \beta(t_i)$. Hence ?*pre* is enabled in $\mathcal{A}_i$. In state 0 for the supervisor, !*pre* is the only possible transition. As the synchronization function f allows $(!pre, \bullet, \cdots, ?pre, \cdots, \bullet)$ the global action pre_i is possible. After this move $\Delta(\mathcal{T})$ reaches state $((1,\mathbf{p_1}),\mathbf{q_1},\mathbf{v_1})$ such that for all $k \in [1..n]$, $\mathbf{q_1}[k] = \mathbf{q}[k], \forall k \neq i$ and $\mathbf{q_1}[i] = \textit{Firing}$. Also $\mathbf{p_1} = \mathbf{p} - {}^\bullet t_i$ and $\mathbf{v_1} = \mathbf{v}$.
Now the only possible transition when the supervisor is in state 1 is an update transition where all the $\mathcal{A}_i$'s synchronize according to f. From $((1,\mathbf{p_1}),\mathbf{q_1},\mathbf{v_1})$ we reach $((2,\mathbf{p_2}),\mathbf{q_2},\mathbf{v_2})$ with $\mathbf{p_2} = \mathbf{p_1}$, $\mathbf{v_2} = \mathbf{v_1}$. For all $k \in [1..n]$, $k \neq i$, $\mathbf{q_2}[k] = t$ *if* $\mathbf{p_1} \geq {}^\bullet t_k$ and $\mathbf{q_2}[k] = \bar{t}$ otherwise. Also $\mathbf{q_2}[i] = \textit{Firing}$. The next global transition must be a $post_i$ transition leading to $((3,\mathbf{p_3}),\mathbf{q_3},\mathbf{v_3})$ with $\mathbf{p_3} = \mathbf{p_2} + t_i^\bullet$, $\mathbf{v_3} = \mathbf{v_2}$ and for all $k \in [1..n]$, $\mathbf{q_3}[k] = \mathbf{q_2}[k], \forall k \neq i$ and $\mathbf{q_3}[i] = \bar{t}$.
From this last state only an update transition leading to $((0,\mathbf{p_4}),\mathbf{q_4},\mathbf{v_4})$ is allowed, with $\mathbf{p_4} = \mathbf{p_3}$, $\mathbf{v_4}$ and $\mathbf{q_4}$ given by: for all $k \in [1..n]$, $\mathbf{q_4}[k] = t$ if $\mathbf{p_3} \geq {}^\bullet t_k$ and $\bar{t}$ otherwise. $\mathbf{v_4}[k] = 0$ if $\mathbf{q_3}[k] = \bar{t}$ and $\mathbf{q_4}[k] = t$ and $\mathbf{v_4}[k] = \mathbf{v_1}[k]$ otherwise. We then just notice that $\mathbf{q_3}[k] = \bar{t}$ iff $\mathbf{p} - {}^\bullet t_i < {}^\bullet t_k$ and $\mathbf{q_4}[k] = t$ iff $\mathbf{p} - {}^\bullet t_i + t_i^\bullet \geq {}^\bullet t_k$. This entails that $\mathbf{v_4}[k] = 0$ iff $\uparrow enabled(t_k, \mathbf{p}, t_i)$ and with (iv) gives $\nu'_k = \mathbf{v_4}[k]$. As $\mathbf{p_4} = \mathbf{p_3} = \mathbf{p_2} + t_i^\bullet = \mathbf{p_1} - {}^\bullet t_i + t_i^\bullet = \mathbf{p} - {}^\bullet t_i + t_i^\bullet$ using (iii) we have $\forall i \in [1..m], M'(p_i) = \mathbf{p_4}[i]$. Hence we conclude that $((0,\mathbf{p_4}),\mathbf{q_4},\mathbf{v_4}) \approx (M',\nu')$.
The converse of statement (2) is straightforward following the same steps as the previous ones.
We now focus on statement (3). According to the semantics of TPNs, a continuous transition $(M,\nu) \xrightarrow{d} (M',\nu')$ is allowed iff $\nu = \nu' + d$ and $\forall k \in [1..n], (M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k))$. As $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$, if $M \geq {}^\bullet t_k$ then $\mathbf{q}[k] = t$ and the continuous evolution for $\mathcal{A}_k$ is constrained by the invariant $x_k \leq \beta(t_k)$. Otherwise $\mathbf{q}[k] = \bar{t}$ and the continuous evolution is unconstrained for $\mathcal{A}_k$. No constraints apply for the supervisor in state 0. Hence the result. □

We can now state a useful corollary which enables us to do TCTL model-checking for TPNs in the next section. We write $\Delta((M,\nu)) = ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$ if $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$, $\Delta(t_i) = pre_i.update.post_i.update$ and also $\Delta(d) = d$. Just notice that Δ is one-to-one and we can use Δ^{-1} as well. Then we extend Δ to transitions by: $\Delta((M,\nu) \xrightarrow{e} (M',\nu')) = \Delta((M,\nu)) \xrightarrow{\Delta(e)} \Delta((M',\nu'))$ with $e \in T \cup \mathbb{R}_{\geq 0}$ (as $\Delta(t_i)$ is a word, this transition is a four step transition in $\Delta(\mathcal{T})$). Again we can extend Δ to runs: if $\rho \in Runs(\mathcal{T})$ we denote $\Delta(\rho)$ the associated run in $Runs(\Delta(\mathcal{T}))$. Notice that Δ^{-1} is only defined for runs σ of $Runs(\Delta(\mathcal{T}))$, the last state of which is of the form $((0,\mathbf{p}),\mathbf{q},\mathbf{v})$ where the supervisor is in state 0. We denote this property $last(\sigma) \models SU.0$.

Corollary 2 $\big(\rho \in Runs(\mathcal{T}) \wedge \sigma = \Delta(\rho)\big)$ *iff* $\big(\sigma \in Runs(\Delta(\mathcal{T})) \wedge last(\sigma) \models SU.0\big)$.

Proof. The proof is a direct consequence of Theorem 5. It suffices to notice that all the finite runs of $\Delta(\mathcal{T})$ are of the form

$$\sigma = (s_0, v_0) \xrightarrow{\delta_1} (s'_0, v'_0) \xrightarrow{w_1} (s_1, v_1) \cdots \xrightarrow{\delta_n} (s'_{n-1}, v'_{n-1}) \xrightarrow{w_n} (s_n, v_n)$$

with $w_i = pre_i.update.post_i.update$, $\delta_i \in \mathbb{R}_{\geq 0}$, and using Theorem 5, if $last(\sigma) \models SU.0$, there exists a corresponding run ρ in $\mathcal{T}$ s.t. $\sigma = \Delta(\rho)$. □

This property will be used in Section 5 when we address the problem of model-checking TCTL for TPNs.

5. Model-Checking of TCTL on Time Petri Nets

In this section we introduce a logic to specify properties of real-time systems and show how we can model-check this logic on bounded TPNs.
We define TCTL (Henzinger et al., 1994) for TPNs. The only difference with the versions of (Henzinger et al., 1994) is that the atomic propositions usually associated with states are now properties of markings. For practical applications with model-checkers, we assume that the TPNs we check are bounded.

Definition 17 (TCTL for TPN) *Assume we have a TPN with n places, and m transitions* $T = \{t_1, t_2, \cdots, t_m\}$. *The temporal logic TPN-TCTL is inductively defined by:*

$$\textit{TPN-TCTL} ::= \mathbf{M} \bowtie \bar{V} \mid \mathbf{false} \mid t_k + c \leq t_j + d \mid \neg\varphi \mid \varphi \rightarrow \psi \mid \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi \mid \varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi \qquad (4)$$

where **M** *and* **false** *are keywords,* $\varphi, \psi \in$ *TPN-TCTL,* $t_k, t_j \in T$, $c, d \in \mathbb{Z}$, $\bar{V} \in (\mathbb{N} \cup \{\infty\})^n$ *and*[6] $\bowtie \in \{<, \leq, =, >, \geq\}$.

Intuitively the meaning of $\mathbf{M} \bowtie \bar{V}$ is that the current marking vector is in relation $\bowtie$ with $\bar{V}$. The meaning of the other operators is the usual one. We use the familiar shorthands:

$$\begin{aligned} \mathbf{true} &= \neg\mathbf{false} \\ \exists\Diamond_{\bowtie c}\phi &= \mathbf{true}\, \exists\mathcal{U}_{\bowtie c}\, \phi \\ \forall\Box_{\bowtie c} &= \neg\exists\Diamond_{\bowtie c}\neg\phi. \end{aligned}$$

The semantics of TPN-TCTL is defined on timed transition systems. Let $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ be a TPN with n places and m transitions and $S_\mathcal{T} = (Q, q_0, T, \rightarrow)$ the semantics of $\mathcal{T}$. Let $\sigma = (s_0, \nu_0) \longrightarrow^{d_1}_{a_1} \cdots \longrightarrow^{d_n}_{a_n} (s_n, \nu_n) \in Runs(\mathcal{T})$. The truth value of a formula φ of TPN-TCTL for a state (M, ν) is given in Fig. 3.
The TPN $\mathcal{T}$ satisfies the formula φ of TPN-TCTL, which is denoted by $\mathcal{T} \models \varphi$, iff the first state of $S_\mathcal{T}$ satisfies φ, i.e., $(M_0, \mathbf{0}) \models \varphi$.
We will see that thanks to Corollary 2, model-checking TPNs amounts to model-checking timed automata.
Let us assume we have to model-check formula φ on a TPN $\mathcal{T}$. Our method consists in using the equivalent timed automaton $\Delta(\mathcal{T})$ defined in Section 4.3. For instance, suppose we want to check $\mathcal{T} \models \forall\Box_{\leq 3}(\mathbf{M} \geq (1, 2))$. The check means that all the states reached within the next

[6] The use of ∞ in $\bar{V}$ allows us to handle comparisons like $M(p_1) \leq 2 \wedge M(p_2) \geq 3$ by writing $\mathbf{M} \leq (2, \infty) \wedge \mathbf{M} \geq (0, 3)$.

$$
\begin{array}{lll}
(M,\nu) \models \mathbf{M} \bowtie \bar{V} & \text{iff} & M \bowtie \bar{V} \\
(M,\nu) \not\models \mathbf{false} & & \\
(M,\nu) \models t_k + c \leq t_j + d & \text{iff} & \nu_k + c \leq \nu_j + d \\
(M,\nu) \models \neg\varphi & \text{iff} & (M,\nu) \not\models \varphi \\
(M,\nu) \models \varphi \rightarrow \psi & \text{iff} & (M,\nu) \models \varphi \text{ implies } (M,\nu) \models \psi \\
(M,\nu) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi & \text{iff} & \exists \sigma \in Runs(\mathcal{T}) \text{ such that:} \\
 & & \quad \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i,\nu_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \text{ and } (s_n,\nu_n) \models \psi \end{cases} \\
(M,\nu) \models \varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi & \text{iff} & \forall \sigma \in Runs(\mathcal{T}) \text{ we have:} \\
 & & \quad \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i,\nu_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \text{ and } (s_n,\nu_n) \models \psi \end{cases}
\end{array}
$$

Figure 3. Semantics of TPN-TCTL

3 time units will have a marking such that p_1 has more than one token and p_2 more than 2. Actually, this is equivalent to checking

$$\forall\Box_{\leq 3}(SU.0 \rightarrow (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$$

on the equivalent timed automaton. Notice that $\exists\Diamond_{\leq 3}(\mathbf{M} \geq (1,2))$ reduces to

$$\exists\Diamond_{\leq 3}(SU.0 \wedge (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$$

We can then define the translation of a formula in TPN-TCTL to standard TCTL for timed automata: we denote TA-TCTL the logic TCTL for timed automata.

Definition 18 (From TPN-TCTL to TA-TCTL) *Let φ be a formula of TPN-TCTL. Then the translation $\Delta(\varphi)$ of φ is inductively defined by:*

$$
\begin{array}{l}
\Delta(\mathbf{M} \bowtie \bar{V}) = \bigwedge_{i=1}^{n} (\mathbf{p}[i] \bowtie \bar{V}_i) \\
\Delta(\mathbf{false}) = \mathbf{false} \\
\Delta(t_k + c \bowtie t_j + d) = x_k + c \bowtie x_j + d \\
\Delta(\neg\varphi) = \neg\Delta(\varphi) \\
\Delta(\varphi \rightarrow \psi) = SU.0 \wedge (\Delta(\varphi) \rightarrow \Delta(\psi)) \\
\Delta(\varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi) = (SU.0 \rightarrow \Delta(\varphi))\, \exists\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \Delta(\psi)) \\
\Delta(\varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi) = (SU.0 \rightarrow \Delta(\varphi))\, \forall\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \Delta(\psi))
\end{array}
$$

$SU.0$ means that the supervisor is in state 0 and the clocks x_k are the ones associated with every transition t_k in the translation scheme.

Theorem 6 *Let $\mathcal{T}$ be a TPN and $\Delta(\mathcal{T})$ the equivalent timed automaton. Let (M,ν) be a state of $S_{\mathcal{T}}$ and $((s,\mathbf{p}),\mathbf{q},\mathbf{v}) = \Delta((M,\nu))$ the equivalent state of $S_{\Delta(\mathcal{T})}$ (i.e. $(M,\nu) \approx ((s,\mathbf{p}),\mathbf{q},\mathbf{v})$). Then $\forall \varphi \in$ TPN-TCTL:*

$$(M,\nu) \models \varphi \textit{ iff } ((s,\mathbf{p}),\mathbf{q},\mathbf{v}) \models \Delta(\varphi).$$

Proof. The proof is done by structural induction on the formula of TPN-TCTL. The cases of $\mathbf{M} \bowtie \bar{V}$, **false**, $t_k + c \leq t_j + d$, $\neg\varphi$ and $\varphi \rightarrow \psi$ are straightforward. We give the full proof for $\varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$ (the same proof can be carried out for $\varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi$).

Only if part.
Assume $(M,\nu) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$. Then by definition, there is a run ρ in $Runs(\mathcal{T})$ s.t. :

$$\rho = (s_0,\nu_0) \longrightarrow^{d_1}_{a_1} (s_1,\nu_1) \cdots \longrightarrow^{d_n}_{a_n} (s_n,\nu_n)$$

and $(s_0,\nu_0) = (M,\nu)$, $\sum_{i=1}^{n} d_i \bowtie c$, $\forall i \in [1..n], \forall d \in [0,d_i), (s_i,\nu_i + d) \models \varphi$ and $(s_n,\nu_n) \models \psi$. With corollary 2, we conclude that there is a run $\sigma = \Delta(\rho)$ in $Runs(S_{\Delta(\mathcal{T})})$ s.t.

$$\sigma = ((l_0,p_0),\bar{q}_0,v_0)) \Longrightarrow^{d_1}_{w_1} ((l_1,p_1),\bar{q}_1,v_1)) \cdots\cdots \Longrightarrow^{d_n}_{w_n} ((l_n,p_n),\bar{q}_n,v_n))$$

and $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$ (this entails that $l_i = 0$.)
Since $(s_n,\nu_n) \approx ((l_n,p_n),\bar{q}_n,v_n))$, using the induction hypothesis on ψ, we can assume that $(s_n,\nu_n) \models \psi$ iff $((l_n,p_n),\bar{q}_n,v_n)) \models \Delta(\psi)$ and thus we can conclude that $((l_n,p_n),\bar{q}_n,v_n)) \models \Delta(\psi)$. Moreover as $l_n = 0$ we have $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0 \wedge \Delta(\psi)$. It remains to prove that all intermediate states satisfy $SU.0 \rightarrow \Delta(\varphi)$. Just notice that all the intermediate states in σ not satisfying $SU.0$ between $((l_i,p_i),\bar{q}_i,v_i)$ and $(((l_{i+1},p_{i+1}),q_{i+1}^{-},v_{i+1}))$ satisfy $SU.0 \rightarrow \Delta(\psi)$. Then we just need to prove that the intermediate states satisfying $SU.0$, i.e., the states $((l_i,p_i),\bar{q}_i,v_i)$ satisfy $\Delta(\varphi)$. As for all $i \in [1..n]$, we have $((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$, with the induction hypothesis on φ, we have $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \models \Delta(\varphi)$. Moreover, again applying theorem 5, we obtain for all $d \in [0,d_i)$: $((l_i,p_i),\bar{q}_i,v_i + d)) \approx (s_i,\nu_i + d)$; applying the induction hypothesis again we conclude that for all $d \in [0,d_i)$ $((l_i,p_i),\bar{q}_i,v_i + d)) \models \Delta(\varphi)$. Hence $((l_0,p_0),\bar{q}_0,v_0)) \models (SU.0 \rightarrow \varphi)\, \exists\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \psi)$.

If part.
Assume $((l_0,p_0),\bar{q}_0,v_0)) \models (SU.0 \rightarrow \Delta(\varphi))\, \exists\mathcal{U}_{\bowtie c}\ (SU.0 \wedge \Delta(\psi))$. Then there is a run

$$\sigma = ((l_0,p_0),\bar{q}_0,v_0)) \Longrightarrow^{d_1}_{w_1} ((l_1,p_1),\bar{q}_1,v_1)) \cdots\cdots \Longrightarrow^{d_n}_{w_n} ((l_n,p_n),\bar{q}_n,v_n))$$

with $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0 \wedge \Delta(\psi)$ and:

$$\forall i \in [1..n], \forall d \in [0,d_i), ((l_i,p_i),\bar{q}_i,v_i)) \models (SU.0 \rightarrow \Delta(\varphi))$$

As $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0$, we can use corollary 2 and we know there exists a run in $Runs(\mathcal{T})$

$$\rho = \Delta^{-1}(\sigma) = (s_0,\nu_0) \rightarrow^{d_1}_{a_1} (s_1,\nu_1) \cdots \rightarrow^{d_n}_{a_n} (s_n,\nu_n)$$

with $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$. The induction hypothesis on $SU.0 \wedge \Delta(\psi)$ and $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0 \wedge \Delta(\psi)$ implies $(s_n,\nu_n) \models \psi$. For all the intermediate states of ρ we also apply the induction hypothesis: each $((l_i,p_i),\bar{q}_i,v_i))$ is equivalent to (s_i,ν_i) and all the states $(s_i,\nu_i + d), d \in [0,d_i)$ satisfy φ. Hence $(s_0,\nu_0) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$. □

Theorem 6 enables to reduce the model-checking of a TPN-TCTL formula φ against a TPN $\mathcal{T}$ i.e., the problem $\mathcal{T} \models \varphi$ to a model-checking of TCTL against TA:

Corollary 3 $\mathcal{T} \models \varphi \iff \Delta(\mathcal{T}) \models \Delta(\varphi)$.

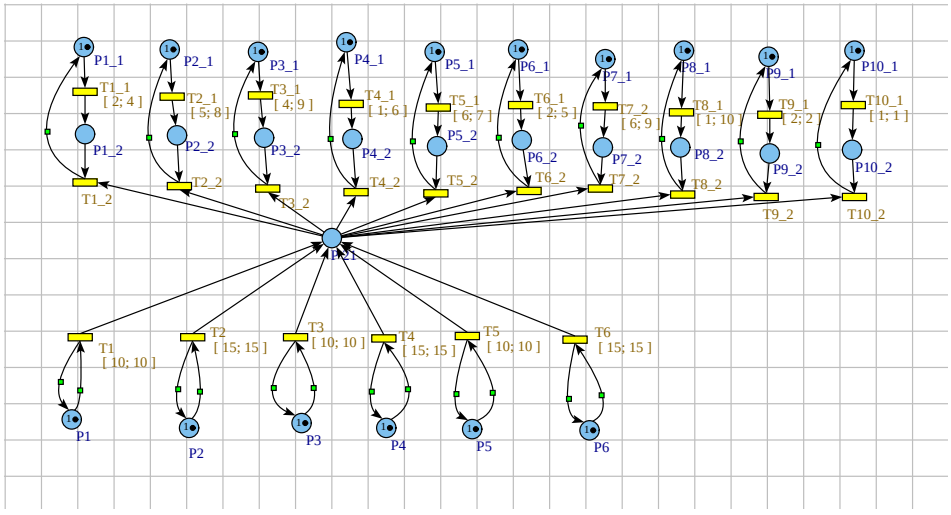


Figure 4. A TPN for a Producer/Consumer example in ROMEO

6. Implementation

In this section, we describe some properties of our translation and important implementation details. Then we report on examples we have checked using our approach and the tool UPPAAL.

6.1. Translation of TPNs to UPPAAL Input Format

The first step in using our approach is to translate an existing TPN into a product of TA. For this we use the TPN tool ROMEO (Gardey et al., 2005) that has been developed for the analysis of TPNs (state space computation and "on-the-fly" model-checking of reachability properties with a zone-based forward method and with the State Class Graph method). ROMEO has a GUI (see Fig. 4) to "draw" Time Petri Nets and an *export to* UPPAAL feature that implements our translation of a TPN into the equivalent TA in UPPAAL input format[7].

The textual input format for TPNs in ROMEO is XML and the timed automaton is given in the ".xta" UPPAAL input format[8]. The translation gives one timed automaton for each transition and one automaton for the supervisor SU as described in Section 4.3. The automata for each transition update an array of integers $M[i]$ (which is the number of tokens[9] in place i in the original TPN). For example, the enabledness and firing conditions of a transition t_i such that ${}^\bullet t_i = (1,0,0)$ and $t_i{}^\bullet = (0,0,1)$, are respectively implemented by $M[0] \geq 1$ and $M[2] := M[2] + 1$. Instead of generating one *template automaton* for each transition, we generate as many templates as types of transitions in the original TPN: the type of a transition is

[7] At least version (3.4.7) of UPPAAL is required to read the files produced by ROMEO.

[8] see `http://www.uppaal.com` for further information about UPPAAL.

[9] The actual meaning of $M[i]$ is given by a table that is available in the ROMEO tool via the "Translate/Indices $\Longrightarrow$ Place/Transition" menu; the table gives the name of the place represented by $M[i]$ as well as the corresponding information for transitions.

the number of input places and output places. For the example of Fig. 4, there are only three types of transitions (one input place to one output place, one to two and two to one) and three templates in the UPPAAL translation. Then one of these templates is instantiated for each transition of the TPN we started with. An example of a UPPAAL template for transitions having one input place and one output place is given in Fig. 5; integers `B1` and `F1` give respectively the index of the unique input place of the transition, and the index of the output place. The timing constraints of the transition are given by `dmin` and `dmax`. We can handle as well transitions with input and output arcs with arbitrary weights (on the examples of Fig. 5 the input and output weights are 1).

In our translation, each transition of the TPN is implemented by a TA with one clock. The synchronized product thus contains as many clocks as the number of transitions of the TPN. At first sight, one can think that the translation we have proposed is far too expensive w.r.t. to the number of clocks to be of any use when using a model-checker like UPPAAL: indeed the model-checking of TA is exponential in the number of clocks. Nevertheless we do not need to keep track of all the clocks as many of them are not useful in many states.

6.2. Inactive Clocks

When a transition in a TPN is disabled, there is no need to store the value of the clock for this transition: this was already used in the seminal paper (Berthomieu and Diaz, 1991). Accordingly when the TA of a transition is in location $\bar{t}$ (i.e., t is not enabled) we do not need to store the value of the clock: this means that many of the clocks can often be disregarded. In UPPAAL, there is a corresponding notion of *inactive clock*:

Definition 19 (UPPAAL Inactive Clock) *Let $\mathcal{A}$ be a timed automaton. Let x be a clock of $\mathcal{A}$ and ℓ be a location of $\mathcal{A}$. If on all paths starting from (ℓ, v) in $S_{\mathcal{A}}$, the clock x is always reset before being tested then the clock x is* inactive *in location ℓ. A clock is* active *if it is not inactive.*

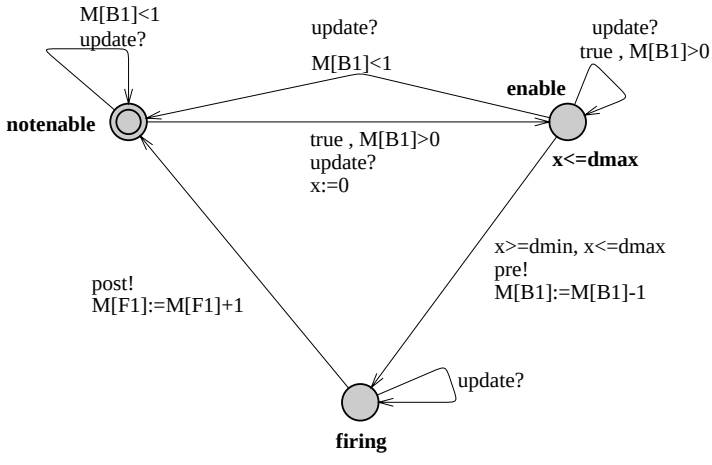


Figure 5. A UPPAAL Template Obtained with the "Export to UPPAAL" Feature of ROMEO

A consequence of the notion of inactive clocks in UPPAAL is that at location ℓ the the constraints on the clocks will only contain the active clocks (they can be omitted in the DBM that represents it). The next proposition (which is easy to prove on the timed automaton of a transition) states that our translation is *effective* w.r.t. active clocks reduction i.e., that when a TA of a transition is not in state t (enabled) the corresponding clock is considered inactive by UPPAAL.

Proposition 1 *Let $\mathcal{A}_i$ be the timed automaton associated with transition t_i of a TPN $\mathcal{T}$ (see Fig. 2, page 240). The clock x_i of $\mathcal{A}_i$ is inactive in locations Firing and $\bar{t}$.*

The recent versions of UPPAAL ($\geq$ 3.4.7) computes active clocks syntactically for each automaton of the product. When the product automaton is computed "on-the-fly" (for verification purposes), the set of active clocks for a product location is simply the union of the set of active clocks of each component. Again without difficulty we obtain the following theorem:

Theorem 7 *Let $\mathcal{T}$ be a TPN and $\Delta(\mathcal{T})$ the equivalent product of timed automata (see section 4.3). Let M be a reachable marking of $S_{\mathcal{T}}$ and ℓ the equivalent[10] location in $S_{\Delta(\mathcal{T})}$. The number of active clocks in ℓ is equal to the number of enabled transitions in the marking M.*

Thanks to this theorem and to the *active clocks reduction* feature of UPPAAL the model-checking of TCTL properties on the network of timed automata given by our translation can be efficient. Of course there are still examples with a huge number of transitions, all enabled at any time that we will not be able to analyze, but those examples cannot be handled by any existing tool for TPN.
In the next subsection we apply our translation to some recent and non trivial examples of TPNs that can be found in (Gardey et al., 2005).

6.3. Tools for Analyzing TPNs

One feature of ROMEO is to export a TPN to UPPAAL or KRONOS but it was originally developed to analyze directly TPNs and has many built-in capabilities: we refer to ROMEO STD for the tool ROMEO with these capabilities (Gardey et al., 2005). TINA (Berthomieu and Vernadat, 2006b) is another state-of-the-art tool to analyze TPNs with some more capabilities than ROMEO STD: it allows to produce an Atomic State Class Graph (ASCG) on which CTL* properties can be checked. Using ROMEO STD or TINA is a matter of taste as both tools give similar results on TPNs.
Table 2 gives a comparison in terms of the classes of property (LTL, CTL, TCTL, Liveness) the tools can handle. The columns UPPAAL and KRONOS in ROMEO give the combined capabilities obtained when using our structural translation and the corresponding (timed) model-checker.
Regarding time performance ROMEO STD and TINA give almost the same results. Moreover with ROMEO STD and TINA, model-checking LTL or CTL properties will usually be faster than using ROMEO +UPPAAL: those tools implement efficient algorithms to produce the (A)SCG needed to perform LTL or CTL model-checking. On one hand it is to be noticed that both ROMEO STD and TINA need 1) to produce a file containing the (A)SCG; and then 2) to run a model-checker on the obtained graph to check for the (LTL, CTL or CTL*) property. This can be prohibitive on very large examples (see (Cassez and Roux, 2006)).
On the other hand neither ROMEO STD nor TINA are able to check quantitative properties such as quantitative liveness (like property of equation (5) below) and TCTL which in general

[10] See Definition 16.

	TINA	ROMEO		
			ROMEO translation from TPN to TA	
		ROMEO STD	UPPAAL	KRONOS
Marking Reachability	Compute marking graph	ROMEO-TCTL[c]	UPPAAL-TCTL[c]	TCTL
LTL	SCG[a] + MC[b]			
CTL	(CTL*) ASCG[a] + MC[b]			
TCTL	–			

[a]SCG = Computation of the State Class Graph ; ASCG = of the atomic SCG.
[b]MC = requires the use of a Model-Checker on the SCG.
[c]Corresponds to a subset of TCTL and a special type of liveness defined by formulas of the form $\forall\Box(\varphi \Longrightarrow \forall\Diamond\Psi)$.

Table 2. What can we do with the different tools and approaches?

cannot be encoded with an observer (when this possible we can translate such a quantitative property into a problem of marking reachability).

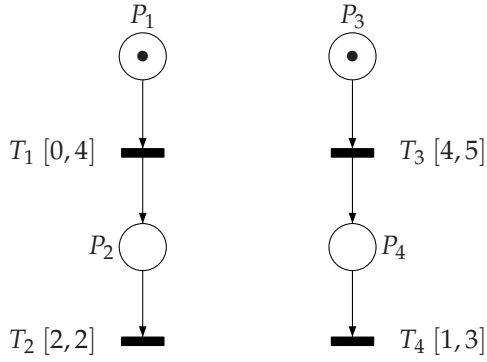


Figure 6. The TPN $\mathcal{T}_g$

Let us consider the TPN $\mathcal{T}_g$ of Fig. 6. The *response* (liveness) property,

$$\forall\Box\big((M[1] > 0 \wedge M[3] > 0 \wedge T_1.x > 3) \Longrightarrow \forall\Diamond(M[2] > 0 \wedge M[4] > 0)\big) \tag{5}$$

where $M[i]$ is the marking of the place P_i, cannot be checked with TINA and can easily be checked with our method using the translation and UPPAAL. This property means that if we do not fire T_1 before 3 t.u. then it is unavoidable that at some point in the future there is a marking with a token in P_2 and in P_4. In UPPAAL we can use the response property template `P -> Q` which corresponds to $\forall\Box(P \Longrightarrow \forall\Diamond Q)$. Using our TPN-TCTL translation we obtain:

```
(SU.0 and  M[1]>0 and M[3]>0 and T_1.x>3) --> (SU.0 and M[2]>0 and M[4]>0)
```

6.4. Experimental Results

We just point out that our translation is syntactic and the time to translate a TPN into an equivalent product of TA is negligible. This is in contrast with the method used in TINA and ROMEO STD where the whole state space has to be computed in order to build some graph (usually very large) and later on, a model-checker has to be used to check the property on the graph. Reports on experimental results on different types of TPNs (cyclic tasks, producers/consumers and large TPNs) can be found in (Cassez and Roux, 2006).

7. Conclusion

In this chapter, we have presented time Petri Nets (TPNs) and a structural translation from TPNs to TA. Any TPN $\mathcal{T}$ and its associated TA $\Delta(\mathcal{T})$ are timed bisimilar.
Such a translation has many theoretical implications. Most of the positive theoretical results on TA carry over to TPNs. The class of TPNs can be extended by allowing strict constraints (open, half-open or closed intervals) to specify the firing dates of the transitions; for this extended class, the following results follow from our translation and from Theorem 5:

- TCTL model checking is decidable for bounded TPNs. Moreover efficient algorithms used in UPPAAL (Pettersson and Larsen, 2000) and KRONOS (Yovine, 1997) are exact for the class of TA obtained with our translation;
- it is decidable whether a TA is non-zeno or not (Henzinger et al., 1994) and thus our result provides a way to decide non-zenoness for bounded TPNs;
- lastly, as our translation is structural, it is possible to use a model-checker to find sufficient conditions of unboundedness of the TPN.

These results enable us to use algorithms and tools developed for TA to check quantitative properties on TPNs. For instance, it is possible to check real-time properties expressed in the logic TCTL on bounded TPNs. The tool ROMEO (Gardey et al., 2005) that has been developed for the analysis of TPN (state space computation and "on-the-fly" model-checking of reachability properties) implements this translation of a TPN into the equivalent TA in UPPAAL input format.
Our approach turns out to be a good alternative to existing methods for verifying TPNs:

- with our translation and UPPAAL we were able to check safety properties on very large TPNs that cannot be handled by other existing tools;
- we also extend the class of properties that can be checked on TPNs to real-time quantitative properties.

Note also that using our translation, we can take advantage of all the features of a tool like UPPAAL: looking for counter examples is usually much faster than checking a safety property. Moreover if a safety property is false, we will obtain a counter example even for unbounded TPNs (if we use breadth-first search).
There are currently new features being developed for tools like ROMEO that enables one to directly check TCTL properties on a TPN without translating it into a TA.

Aknowledgments

The authors wish to thank Didier Lime for his careful reading of this chapter and useful comments to improve many parts of the text.

8. References

Abdulla, P. A. and Nylén, A. (2001). Timed Petri nets and BQOs. In *22nd International Conference on Applications and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70, Newcastle upon Tyne, UK. Springer-Verlag.

Alur, R. and Dill, D. L. (1994). A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235.

Aura, T. and Lilius, J. (2000). A Causal Semantics for Time Petri Nets. *Theoretical Computer Science*, 243(2):409–447.

Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2005a). Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, Uppsala, Sweden. Springer-Verlag.

Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2005b). When are Timed Automata Weakly Timed Bisimilar to Time Petri Nets? In *25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS(05)*, volume 3821 of *Lecture Notes in Computer Science*, Hyderabad, India. Springer-Verlag.

Bérard, B., Cassez, F., Haddad, S., Roux, O., and Lime, D. (2005c). Comparison of Different Semantics for Time Petri Nets. In Xiaoyu, M., Cardoso, J., and Valette, R., editors, *Proceedings of the Third International Symposium on Automated Technology for Verification and Analysis (ATVA'2005)*, volume 3707 of *Lecture Notes in Computer Science*, pages 293–307, Taipei, Taiwan. Springer-Verlag.

Berthomieu, B. and Diaz, M. (1991). Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273.

Berthomieu, B. and Menasche, M. (1983). An Enumerative Approach for Analyzing Time Petri Nets. In Mason, R. E. A., editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, pages 41–46.

Berthomieu, B. and Vernadat, F. (2003). State Class Constructions for Branching Analysis of Time Petri Nets. In *Proc. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 442–457. Springer-Verlag.

Berthomieu, B. and Vernadat, F. (2006a). TINA. http://www.laas.fr/tina.

Berthomieu, B. and Vernadat, F. (2006b). Time Petri Nets Analysis with TINA. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006)*, pages 123–124, Riverside, California, USA. IEEE Computer Society.

Bornot, S., Sifakis, J., and Tripakis, S. (1998). Modeling Urgency in Timed Systems. In de Roever, W. P., Langmaack, H., and Pnueli, A., editors, *International Symposium on Compositionality: The Significant Difference (COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129, Bad Malente, Germany. Springer-Verlag.

Bouyer, P. (2003). Untamable Timed Automata! In *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany. Springer-Verlag.

Bouyer, P. (2004). Forward Analysis of Updatable Timed Automata. *Formal Methods in System Design*, 24(3):281–320.

Bouyer, P., Dufourd, C., Fleury, E., and Petit, A. (2000). Are Timed Automata Updatable? In *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag.

Cassez, F. and Roux, O. H. (2006). Structural Translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata. *The journal of Systems and Software*, 79(10):1456–1468.

Cortès, L. A., Eles, P., and Peng, Z. (2000). Verification of Embedded Systems Using a Petri Net based Representation. In *13th International Symposium on System Synthesis (ISSS 2000)*, pages 149–155, Madrid, Spain.

de Frutos Escrig, D., Ruiz, V. V., and Alonso, O. M. (2000). Decidability of Properties of Timed-Arc Petri Nets. In *21st International Conference on Applications and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, Aarhus, Denmark. Springer-Verlag.

Diaz, M. and Senac, P. (1994). Time Stream Petri Nets: A Model for Timed Multimedia Information. In *15th International Conference on Applications and Theory of Petri Nets (ICATPN'94)*, volume 815 of *Lecture Notes in Computer Science*, pages 219–238, Zaragoza, Spain. Springer-Verlag.

Emerson, E. A. (1990). Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 995–1072. Elsevier.

Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). ROMEO: A Tool for Analyzing Time Petri Nets. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK. Springer-Verlag. http://romeo.rts-software.org.

Gardey, G., Roux, O. H., and Roux, O. F. (2003). Using Zone Graph Method for Computing the State Space of a Time Petri Net. In *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, Marseille, France. Springer-Verlag.

Gardey, G., Roux, O. H., and Roux, O. F. (2006). State Space Computation and Analysis of Time Petri Nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 6(3):301–320.

Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1994). Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244.

Jones, N. D., Landweber, L. H., and Lien, Y. E. (1977). Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299.

Khansa, W., Denat, J.-P., and Collart-Dutilleul, S. (1996). P-Time Petri Nets for Manufacturing Systems. In *International Workshop on Discrete Event Systems (WODES'96)*, pages 94–102, England. IEEE Computer Society.

Laroussinie, F. and Larsen, K. G. (1998). CMC: A Tool for Compositional Model-Checking of Real-Time Systems. In Budkowski, S., Cavalli, A. R., and Najm, E., editors, *Proceedings of IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques*

for Distributed Systems and Communication Protocols (FORTE'XI) and Protocol Specification, Testing and Verification (PSTV'XVIII), volume 135 of *IFIP Conference Proceedings*, pages 439–456, Paris, France. Kluwer Academic Publishers.

Larsen, K. G., Pettersson, P., and Yi, W. (1997). UPPAAL in a Nutshell. *International Journal of Software Tools for Technology Transfer*, 1(1–2):134–152. http://www.uppaal.com/.

Lilius, J. (1998). Efficient State Space Search for Time Petri Nets. *Electronic Notes in Theoretical Computer Science*, 18.

Lime, D. and Roux, O. H. (2006). Model Checking of Time Petri Nets Using the State Class Timed Automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications*, 16(2):179–205.

Merlin, P. M. (1974). *A Study of the Recoverability of Computing Systems*. PhD thesis, Dep. of Information and Computer Science, Univ. of California, Irvine, CA.

Pettersson, P. and Larsen, K. G. (2000). UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44.

Pezzè, M. (1999). Time Petri Nets: A Primer Introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain.

Popova, L. (1991). On Time Petri Nets. *Journal of Information Processing and Cybernetics, EIK*, 27(4):227–244.

Ramchandani, C. (1974). *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA. Project MAC Report MAC-TR-120.

Rokicki, T. G. (1993). *Representing and Modeling Circuits*. PhD thesis, Stanford University.

Sava, A. T. (2001). *Sur la synthèse de la commande des systèmes à évènements discrets temporisés*. PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France.

Sifakis, J. (1980). Performance Evaluation of Systems using Nets. In Brauer, W., editor, *Net theory and applications : Proceedings of the advanced course on general net theory, processes and systems*, volume 84 of *Lecture Notes in Computer Science*, pages 307–319, Hamburg, Germany. Springer-Verlag.

Sifakis, J. and Yovine, S. (1996). Compositional specification of timed systems. In Puech, C. and Reischuk, R., editors, *13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359, Grenoble, France. Springer-Verlag.

Tripakis, S. (1999). Timed Diagnostics for Reachability Properties. In Cleaveland, R., editor, *Proc. 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 59–73, Amsterdam, The Netherlands. Springer-Verlag.

Yovine, S. (1997). KRONOS: A Verification Tool for Real-Time Systems. *International Journal of Software Tools for Technology Transfer*, 1(1–2):123–133.

12

Timed Hierarchical Object-Oriented Petri Net

Hua Xu
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084,
P. R. China

1. Introduction

Petri nets (Murata, 1989) (Peterson, 1991) have been widely used to model various discrete event systems (Moody & Antsaklis, 1998). Characterized as concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic (Murata, 1989), Petri nets have gained more and more applications. However, when they are used to analyze and model systems of different domains, the shortages of this kind of formal method still exist. Basic Petri nets lack temporal knowledge description, so they have failed to describe the temporal constraints in time critical or time dependent systems. The introduction of temporal knowledge into Petri nets has increased not only the modeling power but also the model complexity (Wang et al. 2000). The improved models of Petri nets (Wang, 1998) include Timed Petri Net (Ramchandi, 1974), Stochastic Timed Petri Net (Florin etc al., 1991) and Time Petri Net (TPNs) (Merlin & Farber, 1976). In TPNs (Merlin & Farber, 1976), each bar has two times specified. The first time denotes the minimal time that must elapse from the time that all the input conditions of a bar are enabled until this bar can fire. The other time denotes the maximum time that the input conditions can be enabled and the bar does not fire. After this time, the bar must fire. In general, these two times give some measures of minimal and maximal execution times of the bars.

The reachability (coverability) analysis is one of the main analysis methods for Petri nets (Murata, 1989), in which the coverability tree is always used. It permits the automatic translation of behavioral specification models into a state transition graph made up of a set of states, a set of actions, and a succession relation associating states through actions (Bucci & Vivario, 1995). That is to say, it involves essentially the enumeration of all reachable markings or their coverable markings. This representation makes such properties as deadlock and reachability (Zhou, 1995) explicit, and allows the automatic verification of ordering relationships among task execution times (Tsai et al., 1995).

Although the reachability analysis method can be used for all nets, it is only limited to "small" nets due to the complexity of the state-space explosion. The same thing also happens in the analysis of TPN models. Sloan et al. (Sloan & Buy, 1996) developed several reduction rules for TPN analysis that work at an individual transition level. These reduction rules help to reduce the complexity of TPN analysis to some extent. However, it is not a trivial work to automatically search the preconditions of applying these reduction rules for a

complex TPN. Wang et al. proposed the compositional time Petri nets and the corresponding component-level reduction rules (Wang et al. 2000). Each of the reduction rules transforms a TPN component to a small one while maintaining the net's external observable timing properties. The application of these rules will dramatically reduce the size of a TPN. However, all of the methods or models only reduce the complexity after the model becomes complex. It can not avoid the complexity to the best of its ability according to the analysis requirements when it is modeled.

These years, the usefulness of the object-oriented concepts has been recognized, because it allows us to describe systems easily, intuitively and naturally. These years, the object-oriented formal methods such as object Petri nets (OPN) (Bastide, 1995), VDM++ (Harel & Gery , 1996), Object-Z (Schuman, 1997), etc are suggested. Among the studies, the research on OPN has been focused on the extending Petri net formalism to OPN such as HOONet (Hong & Bae, 2000), OBJSA (Battiston et al. 1988), COOPN/2 (Biberstein & Buchs, 1994) and LOOPN++ (Lakos & Keen, 1994), which are suggested on the base of colored Petri Net (CPN) (Jensen, 1992). Object-oriented Petri net (OPN) can model different systems easily, intuitively and naturally. Abstraction is one of OPN characters compared with basic Petri nets. OPN can model various systems hierarchically and the models can be analyzed even if they have not been completed. So the complexity of OPN models can be simplified at the beginning of modeling stage according to the analysis requirements. Although the results of such studies have shown promise, these nets do not fully support time critical (time dependent) system modeling and analysis, which may be complex, midsize or even small. When time critical systems with any sizes are modeled, it requires formal modeling and analysis method to support temporal description and object-oriented concepts. That is to say, TPN and OPN need to be combined.

Firstly, this chapter formally proposes a high-level Petri net called timed hierarchical object-oriented Petri net (TOPN) (Xu & Jia, 2006) (Xu & Jia, 2005-2), which supports not only temporal description but also OO concepts. On one hand, TOPN has extended a model of Object-Oriented Petri Nets to allow modeling and analyzing complex time critical systems. Modeling features in TOPN support abstracting complex systems, so the corresponding models can be simplified effectively. In the proposed TOPN, a duration is also attached to each object accounting for the minimal and maximal amount of time between which that the behavior of the object can be completed once fired. On the other hand, this chapter also addresses the problem of the state analysis of TOPN models, what makes it possible to judge the model consistency at a given moment of time. On the base of Yao's extended state graph (ESG) (Yao, 1994), TOPN extended state graph (TESG) is presented for incremental reachability analysis for temporal behavior analysis. In particular, a new way is investigated to represent and deal with the objects with temporal knowledge.

Secondly, in order to extend a model of TOPN to allow modeling and analyzing dynamic systems with timing effect on system information, fuzzy concept is introduced into TOPN and fuzzy timed object-oriented Petri net (FTOPN) (Xu & Jia, 2005-1) is proposed. Temporal fuzzy sets are attached to each transition objects in TOPN accounting for the aging of information. In particular, a new way is investigated to represent and deal with timing effect in dynamic systems. FTOPN also supports learning similar to that in fuzzy timed Petri net (Pedryz & Camargo, 2007). FTOPN is also used to model a real decision making procedure of one cooperative multiple robot system (CMRS) to demonstrate its following benefits:

independent training for its supporting object abstraction and size reconfiguration for its object granularity control function.

Finally, in order to model CMRS, a CMRS modeling method called fuzzy timed agent based Petri nets (FTAPN) (Xu & Jia, 2007) is proposed on the base of FTOPN, because it can be regarded as a kind of multi-agent system (MAS) and the agent is also a special kind of object. FTAPN can be used to model and illustrate both the structural and dynamic aspects of CMRS. Supervised learning is supported in FTAPN. As a special type of high-level object, agent is introduced, which is used as a common modeling object in FTAPN models. The proposed FTAPN can not only be used to model CMRS and represent system aging effect, but also be refined into the object-oriented implementation easily. At the same time, it can also be regarded as a conceptual and practical artificial intelligence (AI) tool for multi-agent system (MAS) into the mainstream practice of software development.

This chapter has just been arranged as the following. Section 1 makes a quick review of the relative study of Petri Nets. In section 2 of this chapter, it justifies the need for defining TOPN through interpreting how to combine the time restricting information with HOONet. An informal and intuitive behavior semantics of TOPN has been introduced in section 3. Then, in section 4, the constructing algorithm of reachability tree is presented, which can support most of the property analysis of TOPN. In section 5, FTOPN is proposed on the base of TOPN, and FTOPN has been used to model and analyze the decision procedure of one CMRS. Then, FTAPN is presented on the base of FTOPN and it is used to model and analyze one CMRS to demonstrate its effectiveness in section 6. Section 7 concludes the work in this chapter and suggests further research issues in the future.

2. The basic concepts of TOPN

In this section, some important basic concepts of Petri nets are firstly reviewed. Then the definitions of TOPN are presented. At the same time, the enabling rules and the firing rules of TOPN are presented.

2.1 A brief review of basic Petri nets

In this subsection, we will quickly review some key definitions. A more general discussion on Petri nets can be found in Peterson's book (Peterson, 1991) and in the excellent survey article by Murata (Murata, 1989).

A Petri net is a five-tuple PN= (P, T, F, W, M_0) where P and T are the node sets and F is the edge set of a directed bipartite graph, and M_0: P→N is called the initial marking (or initial state) of PN. (We use N to denote the set {0, 1, 2…}.) We call P the set of places of PN and T the set of transitions of PN. In diagrams, we will show places as circles and transitions as bars. Formally, $F \subseteq (P \times T) \cup (T \times P)$ and F is called the flow relation (or edges) of PN. W: F → {1, 2, 3…} and it is called the weight of a flow. In general, a marking of PN associates a nonnegative integer number of markers or tokens with each place.

For net PN= (P, T, F, W, M_0), we use the following symbols and notations for the sets of predecessors and successors of a place p∈P and transition t∈T.

- •t={p | (p, t)∈F} =the set of input places of t,
- t•={p | (t, p) ∈F}=the set of output places of t,

- $\bullet p=\{t \mid (t, p)\in F\}$=the set of input transitions of p,
- $p\bullet=\{t \mid (p, t)\in F\}$ =the set of output transitions of p.

A transition is enabled when all its input places have at least one token. When an enabled transition t is fired, a token is removed from each input place of t and a token is added to each output place; this gives a new marking. For net PN= (P, T, F, W, M_0), the language of PN, denoted as L (PN), is the set of all legal sequences $\omega\in T^*$ of transition firings starting from marking M_0.

Petri net PN= (P, T, F, W, M_0) is safe if $M_0 : P\rightarrow\{0, 1\}$, and if all markings reachable by legal sequences of transition firings from the initial marking have either zero or more tokens in every place.

2.2 High-level Petri nets

There are different definitions and terminology of TPN and OPN. In this chapter, our work is based on the Merlin's TPN and Hong's OPN which is called HOONet (Hong & Bae, 2000). A time Petri net is also a tuple *TPN*=(*PN*, *SI*). PN is a basic Petri net. And SI is a mapping called a static interval, SI: $T\rightarrow Q^* \times (Q^*\cup\infty)$, where Q^* is a set of nonnegative rational numbers.

HOONet is a high-level Petri net supporting the representation scheme of object-oriented concepts. A HOONet model is represented as Petri-net form for an object and has components to represent a unique name, attributes and its behaviors (methods) of an object.

Definition 1: HOONet is defined with a tuple HOONet= (OIP, ION, DD), where

1. OIP (object identification place) is a special place which is defined as a tuple, OIP=(*oip,pid,*M_0*,status*), where
   - *oip* is a unique name of a HOONet model.
   - *pid* is a unique process identifier that distinguishes the multiple instances of an object.
   - M_0 is an initial marking function.
   - *status* is a flag variable (either pre value or post value) to represent the specific states of OIP.
2. ION (internal object net) is a variant of CPN (colored Petri nets) that represents the internal behaviors of an object, which is defined as a tuple ION=(*P,T,A,C,N,G,E,F,*M_0), where
   - *P, T* and *A* are finite sets of places, transitions and arcs respectively.
   - *C, N, G* and *E* mean the functions of a color set, a node, a guard and an arc expression, respectively. They are the same as defined in (Jensen, 1992).
   - *F* is a special arc from transitions to OIP, and depicted as those a rim of ION, and
   - M_0 is a function giving initial marking to specific places.
3. DD (data dictionary) contains the declarations of variable, token types, and functions per a HOONet model using standard CPN ML (Jensen, 1992). □

Definition 2: A set of place types in HOONet, P=(P_i,P_a), where

1. Primitive place P_i is a basic type of places that represent the local states of a system, the same as in general CPN (Jensen, 1992).

2. Abstract place P_a= (*pn, refine_state, action*) is a place type which represents abstract states, where
    - *pn* is the name of an abstract place.
    - *refine_state* is a flag variable denoting the refinement of an abstract place.
    - *action* is a static reaction that imitates the internal behaviors of an abstract place. □

Definition 3: A set of transition types in HOONet, T={T_i, T_a, T_c},where
1. Primitive transition T_i is a basic transition type in general CPN
2. Abstract transition T_a = (tn, refine_state, action), where
    - tn is the name of an abstract transition.
    - refine_sate and action have the same meanings as in the definition of the abstract place.
3. Communicative transition T_c =(tn, target, ctype, action) is a transition type that represents calling a method, where
    - tn is the name of a communicative transition.
    - target is a flag variable denoting whether the method called from Tc, is modeled (a "yes" value) or not (a "no" value).
    - ctype is also a flag variable denoting whether the interaction of Tc is synchronous (a "SYNC" value) or asynshronous (an "ASYN" value).
    - action is the static reaction that reflects the execution results of the called method.

The variable *ctype* with its "SYNC" value denotes that the caller waits for the result from the called method. With "ASYN" value, the token is duplicated. Each of the duplicated tokens is transferred to the called object and the next place in its net, respectively.

2.3 Timed hierarchical object-oriented Petri net

The purpose of designing timed hierarchical object-oriented Petri net (TOPN) is to aid in the modeling and analysis of real time systems and bridge the gap between the formal treatment of object-oriented Petri nets and temporal reduction approach for the modeling, analysis, and prototyping of complex time critical systems.

A TOPN model is a variant HOONet representation that corresponds to the class with temporal property in object-oriented paradigm. Like the HOONet, TOPN is composed of four parts: object identification place (OIP) is a unique identifier of a class; internal timed object net (ION) is a net to depict the behaviors (methods) of a class; data dictionary (DD) declares the attributes of a class in TOPN; and static time interval function (SI) binds the temporal knowledge of a class in TOPN.

Definition 4: TOPN is a four-tuple: TOPN= P (OIP, ION, DD, SI), where:
1. OIP=(oip, pid, M_0, status), oip, pid, M_0 and status are the same as those in HOONet.
    - oip is a variable for the unique name of a TOPN.
    - pid is a unique process identifier to distinguish multiple instances of a class, which contains return address.
    - M_0 is the function that gives initial token distributions of this specific value to OIP.
    - status is a flag variable to specify the state of OIP.

2. ION is the internal net structure of TOPN to be defined in the following. It is a variant CPN that describes the changes in the values of attributes and the behaviors of methods in TOPN.
3. DD formally defines the variables, token types and functions (methods) just like those in HOONet (Hong & Bae, 2000).
4. SI is a static time interval binding function, SI: {OIP}→Q*, where Q* is a set of time intervals.□

According to the *definition 4*, the general structure of TOPN is shown in Fig.1. In time critical systems, time relates to events. While in Petri net, events occur and originate from system behaviors. And system behaviors stem from the behavior properties of objects in TOPN. These objects include transitions, abstract places and other TOPN objects. So not only transitions, but also all TOPN objects including abstract places, etc need to be restricted by time condition.

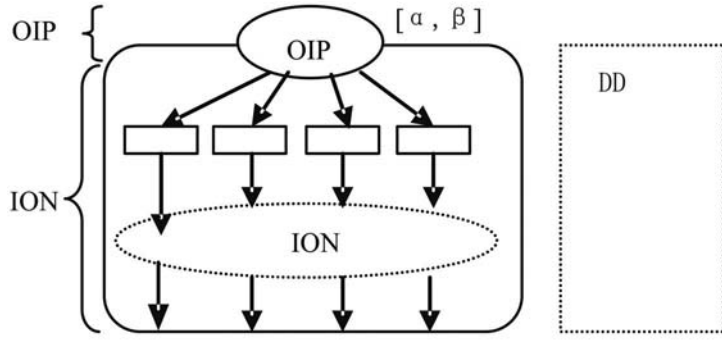


Fig. 1. The General Structure of TOPN

An event in a time critical system can be thought of as an interval [s, t] on the time line where s is its starting endpoint and t is its terminating endpoint, having a duration given by t-s≥0. The special case of time interval where t=s is a *point event*. Otherwise, it is an *interval event*. In the corresponding time interval [s, t] of event firing, s is the earliest firing time (EFT) and t is the latest firing time (LFT). In the changes of TOPN behavior, events are regarded as interval events. The temporal knowledge in TOPN is represented as time intervals.

Similar to HOONet, TOPN is also a kind of hierarchical net. In TOPN, the whole TOPN model is also an object, and it is always regarded as an abstract place object. Its realizing details are depicted in ION. Inside the ION, abstract objects may also be included. The realizing details of these objects can also be depicted as a TOPN. The definition of ION is just like the following.

Definition 5: An internal object net structure of TOPN, ION = (P,T,A,K,N,G,E,F,M_0)

1. P and T are finite sets of places and transitions with time restricting conditions attached respectively.
2. A is a finite set of arcs such that P∩T=P∩A=T∩A=Φ.
3. K is a function mapping from P to a set of token types declared in DD.

4. N, G, and E mean the functions of nodes, guards, and arc expressions, respectively. The results of these functions are the additional condition to restrict the firing of transitions. So they are also called additional restricting conditions.
5. F is a special arc from any transitions to OIP, and notated as a body frame of ION.
6. M_0 is a function giving an initial marking to any place the same as those in HOONet (Hong & Bae, 2000). □

Similar to common OPNs, basic OPN components and additional restricting conditions are included in the detailed ION structure. The basic OPN components may include common components (transition and place) and abstract components. If the model needs to be analyzed in details, the abstract components in ION should be refined. At the same time, the ION is unfolded. The following definitions of abstract components in TOPN are the base of refining abstract component. The abstract components in TOPN include timed abstract transitions, timed abstract communication transitions and timed abstract places.

Definition 6: A set of places in TOPN is defined as P=PIP∪TABP, where

1. PIP is the set of primitive places similar to those in PNs (Murata, 1989) (Peterson, 1991).
2. Timed abstract place (TABP) is a four-tuple: TABP= TABP(pn_{TABP}, refine $state_{TABP}$, $action_{TABP}$, SI_{TABP}), where
   - pn_{TABP} is the identifier of the abstract timed place.
   - refine $state_{TABP}$ is a flag variable denoting whether this abstract place has been refined or not.
   - $action_{TABP}$ is the static reaction imitating the internal behavior of this abstract place.
   - SI_{TABP} is also a static time interval binding function from a set of TABPs to a set of static time intervals. □

There are two kinds of places in TOPN. They are common places (represented as circles with thin prim) and abstract places (represented as circles with bold prim) described in Fig.2. Abstract places are also associated with a static time interval. Because at this situation, abstract places represent not only firing conditions, but also the objects with their own behaviors. So, abstract places in TOPN also need to be associated with time intervals.

Generally, the abstract place—TABP is always represented as a kind of abstract form in higher layers in TOPN models. At this time, "refine state" dedicates that it is in abstract form. However, if "refine state" denotes that it is in the refined state, TABP will have been refined into the corresponding TOPN which is defined in lower layers.

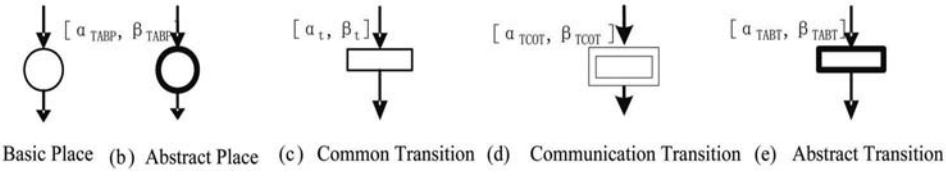


Fig. 2. Places and Transitions in TOPN

Definition 7: A set of transitions in TOPN can be defined as T= TPIT∪TABT∪TCOT, where

1. Timed primitive transition TPIT = TPIT (BAT, SI_{TPIT}), where
   - BAT is the set of common transitions.

- SI_{TPIT} is a static time interval binding function, SI: {TPIT}→Q*, where Q* is a set of time intervals.

4. Timed abstract transition TABT= TABT (tn_{TABT}, refine $state_{TABT}$, $action_{TABT}$, SI_{TABT}), where
   - tn_{TABT} is the name of this TABT.
   - refine $state_{TABT}$ is a flag variable denotes whether this TABT has been refined or not.
   - $action_{TABT}$ is the static reaction imitating the internal behavior of the TABT.
   - SI_{TABT} is a static time interval binding function, SI: {TABT}→Q*, where Q* is a set of time intervals.
5. Timed communication transition TCOT=TCOT (Tn_{TCOT}, $target_{TCOT}$, comm $type_{TCOT}$, $action_{TCOT}$, SI_{TCOT}).
   - Tn_{TCOT} is the name of TCOT.
   - $target_{TCOT}$ is a flag variable denoting whether the behavior of this TCOT has been modeled or not. If $target_{TCOT}$ ="Yes", it has been modeled. Otherwise, if $target_{TCOT}$ ="No", it has not been modeled yet.
   - comm $type_{TCOT}$ is a flag variable denoting the communication type. If comm $type_{TCOT}$ ="SYNC", then the communication transition is synchronous one. Otherwise, if comm $type_{TCOT}$ ="ASYN", it is an asynchronous communication transition.
   - $action_{TCOT}$ is the static reaction imitating the internal behavior of this TCOT.
   - SI_{TCOT} is a static time interval binding function, SI: {TCOT}→Q*, where Q* is a set of time intervals. □

Just like those in HOONet, there are three kinds of transitions in TOPN. The timed primitive transition (represented as rectangles with thin prim), timed abstract transition (represented as rectangles with bold prim) and timed communication transition (represented as rectangles with double thin prim). They are depicted in Fig.2. Different transitions represent different system behaviors. So, temporal intervals are associated with all of these transitions in TOPN. Abstract transitions are also TOPN objects. They can be refined in lower layers.

The definition of abstract transitions mentioned above is also a kind of abstract form in higher layers of TOPN models, when "refine state" indicates it is in the abstract form. While, if "refine state" denotes it is in refined state, the corresponding abstract transition should be refined into the corresponding TOPN which is defined in lower layers.

From the definitions mentioned above, TOPN are hierarchical just like the structure of object models. In the higher levels of the model, its components may be in abstract form and the model is simple. In the unfolded model where the abstract components are refined, the TOPN model may be complex, but the realizing details are clear. So, according to the analysis requirements, users can analyze the TOPN models in different layers, even if the detailed realization in lower layers have not been completed yet.

3. Behavior semantics of TOPN

3.1 Execution paths

State changes relate to the events in TOPN. However, events may stem from transition firing or TABP behaviors. The state changes in TOPN relate to the schedule and the associated

temporal interval tightly. In order to analyze the dynamics of TOPN, the definition of schedule and path is given in the following.

Definition 8: In Petri net N, if the state Mn is reachable from the initial state M_0, then there exists a sequence of fired transitions from M_0 to Mn. This sequence is called a path or a schedule ω from M_0 to Mn. It can be represented as:

$$\text{Path} = \{M_0,t_1,M_1,\ldots,t_n,M_n\} \text{ or } \omega = \{M_0,t_1,M_1,\ldots,t_n,M_n\}$$

$$t_i \in N.T;\ 1 \le i \le n$$

And the schedule set of Petri net N with initial marking M_0 is represented as $L(N,M_0)$. □

Just like those in TPN (Merlin & Farber, 1976) (Harel & Gery, 1996), if the number of solid tokens residing in the input place equals or exceeds the weight of the input arc, the forward transition is enabled. However, when one TABP is marked by enough hollow tokens compared with the weight of internal arcs in its refined TOPN, it is also enabled at this time. After its internal behaviors have completed, the color of tokens residing in it become from hollow to solid, which are similar to those in common places. So TABPs also manifest actions in TOPN. An extended definition of path in TOPN is given in the following, in which TABP is extended into the schedule.

Definition 9: If the state Mn is reachable from the initial state M_0, then there exists a sequence of marked abstract places and fired transitions from M_0 to Mn. This sequence is called a path or a schedule ω from M_0 to Mn. It can be represented as:

$$\text{Path} = \{PA_1, PA_2, \ldots, PA_n\} \text{ or } \omega = \{PA_1, PA_2, \ldots, PA_n\}$$

where $PA_i \in T \cup TABP$ and $1 \le i \le n$.

Definition 10: Let t be a TOPN transition and let $\{PA_1, PA_2, \ldots, PA_n\}$ be a path, add t_i into the path is expressed as $\{PA_1, PA_2, \ldots, PA_n\} + t = \{ PA_1, PA_2, \ldots, PA_n, t\}$.

Let p be an abstract place and let $\{ PA_1, PA_2, \ldots, PA_n\}$ be a path, add p into the path is expressed as $\{ PA_1, PA_2, \ldots, PA_n\} + p = \{ PA_1, PA_2, \ldots, PA_n, p\}$, where $PA_i \in T \cup TABP$ and $1 \le i \le n$.

Definition *11*: For a TOPN N with schedule ω, we denote the state reached by starting in N's initial state and firing each transition in ω at its associated time φ(N,ω). The time of φ(N,ω) is the global firing time of the last transition in ω .

When the relative time belongs to the time interval attached to the transition or the TABP and the corresponding object is also enabled, then it can be fired. If a transition has been fired, the marking may change like that in PN (Wang, 1998). If a TABP is fired, then the hollow token(s) change into solid token(s), and the tokens still reside in the primary place. At this time, the new relative time intervals of every object are calculated like those in (Harel & Gery, 1996).

3.2 Enabling rules and firing rules

State changes in TOPN stem from the behavior executions in TOPN. The execution of a TOPN depends on two main factors. Firstly, it is the number and distribution of tokens in

the TOPN. Tokens reside in the places and control the execution of the transition. Secondly, its execution depends on the definition of execution time represented as time intervals. A TOPN executes by firing transitions.

The dynamic behavior can be studied by analyzing the distribution of tokens (markings) in TOPN. So the enabling rule and firing rule of a transition in TOPN are introduced in the following, which govern the flow of tokens.

Enabling Rule:

1. A transition t in TOPN is said to be enabled if each input place p of t contains at least the number of solid tokens equal to the weight of the directed arcs connecting p to t:
   $M(p) \geq I(t, p)$ for any p in P, the same as in ordinary Petri nets, where M(p) is the marking of the place p and I(t, p) is the weight of the input arc from the place p to the transition t.
2. If the place is TABP, it will be marked with a hollow token and TABP is enabled. At this time, the ION of the TABP is enabled. After the ION is executed, the tokens in TABP are changed into solid ones. □

Firing Rule:

1. For a transition:
   a. An enabled transition in TOPN may or may not fire depending on the additional interpretation (Merlin & Farber, 1976) (Bucci & Vivario, 1995) (Harel & Gery, 1996), and
   b. The relative time θ, relative to the absolute enabling time τ, is not smaller than the earliest firing time (EFT) of transition t_i, and not greater than the smallest of the latest firing time (LFT) of all the transitions enabled by marking M (Hong & Bae, 2000):
      $$\text{EFT of } t_i \leq \theta \leq \min(\text{LFT of } t_k)$$
      where k ranges over the set of transitions enabled by M, the same as (Hong & Bae, 2000).
   c. After a transition t_i (common one or abstract one) in TOPN is fired at a time θ, TOPN changes to a new state. The new states can be computed as the following:

- The new marking M' (token distributions) can be computed as the following:
  If the output place of t_i is TABP,
  then $M'(p) = \text{attach}(*, (M(p)-I(t_i,p)+O(t_i,p)))$;
  else $M'(p)=M(p)-I(t_i,p)+O(t_i,p)$;
  The symbol "*" attached to the markings of TABP represents as hollow tokens in TABP.
- The computation of the new firing interval I' is the same as those in (Harel & Gery, 1996), (Yao, 1994), as
  $$I'=(\max(0,EFT_k-\theta_k) , (LFT_k-\theta_k))$$
  where EFT_k and LFT_k represents the lower and upper bound of interval in I corresponding to t_k in TOPN, respectively.
- The new path can be computed as path' = path + t_i .

2. For a TABP
   a. The relative time θ should satisfy the following conditions:
      $$\text{EFT of } t_i \leq \theta \leq \min(\text{LFT of } t_k)$$

where t_k belongs to the set of the places and transitions which have been enabled by M.

b. After a TABP p in TOPN is executed at a time θ, TOPN states change. The new marking can be computed as the following.

- The new markings are changed for the corresponding TABP p, as
  M′(p)= remove_attach (*, M(p))
  The symbol "*" is removed from the marking of TABP. Then the marking is the same as those of common places. The change represents that the internal actions of TABP have been finished. Tokens of TABP have been changed into solid ones.
  To compute the new time intervals is the same as that mentioned above.
- The new path can be decided by path′ = path + p. □

When the number of tokens satisfies the conditions of enabling rule, the corresponding transitions or TABPs are enabled. Only if the corresponding objects are enabled and the relative time is in the time interval, can the objects be fired. The relative firing time may be stochastic, but it is after EFT and before LFT. In TOPN, the firing procedures are considered to be instantaneous and their execution delay can be considered in the time interval of execution conditions.

4. Reachability analysis

4.1 Analysis algorithm

The purpose of TOPN is to aid in modeling and analysis of complex time critical systems. From the point of TOPN definition, TOPN can describe the temporal constraints in time critical systems. Then the model analysis method especially reachability analysis, need to be discussed. In order to analyze TPN (Yao, 1994) models, Yao has presented extended state graph (ESG) to analyze TPN models. On the base of ESG, an extended TOPN state graph has been presented in this section, into which temporal reasoning has also been introduced.

In a TOPN model, an extended state representation "ES" is 3-tuple, where ES=(M, I, path) consisting of a marking M, a firing interval vector I and an execution path. According to the initial marking M_0 and the firing rules mentioned above, the following marking at any time can be calculated. The vector--"I" is composed of the temporal intervals of enabled transitions and TABPs, which are to be fired in the following state. The dimension of I equals to the number of enabled transitions and TABPs at the current state. The firing interval of every enabled transition or TABP can be got according to the formula of I′.

Definition 12: A TOPN extended state graph (TESG) is a directed graph. In TESG, nodes represent TOPN model states. In TESG, there is an initial node, which represents the TOPN model initial state. Arcs denote the events, which make model state change. There are two kinds of arcs from one state ES to another one ES′ in TESG.

1. The state change from ES to ES′ stems from the firing of the transition t_i. Correspondingly, there is a directed arc from ES to ES′, which is marked by t_i.
2. If the internal behavior of the TABP—"p_i" makes the TOPN model state change from ES to ES′, then in TESG there is also a directed arc from ES to ES′. It is marked by p_i. □

On the base of Petri net analysis method (PN and TPN) and the definition of TESG, the TESG of one TOPN model can be constructed by the following step:

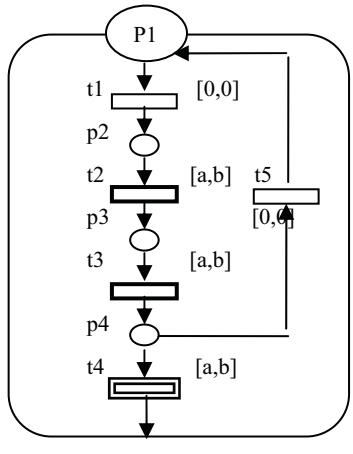
Step 1) Use the initial state ES_1 as the beginning node of TESG, where $ES_1=(M_0, [0,0], \Phi)$.

Step 2) Mark the initial state "New".

Step 3) While (there exist nodes marked with "new") do
Step 3.1) Choose a state marked with "new".
Step 3.2) According to the enabling rule, find the enabled TOPN objects at the current state and mark them "enabled".
Step 3.3) While (there exist objects marked with "enabled") do
Step 3.3.1) Choose an object marked with "enabled".
Step 3.3.2) Fire this object and get the new state ES_2.
Step 3.3.3) Mark the fired object "fired" and mark the new state ES_2 "new".
Step 3.3.4) Draw a directed arc from the current state ES_1 to the new state ES_2 and mark the arc with the name of the fired object and relative firing temporal constraint.
// The internal "while" cycle ends.
Step 3.4) Mark the state ES_1 with "old".
// The external "while" cycle ends.

TESG describes state changes in TOPN models. In TESG, not only state changing sequence, but also dynamic temporal constraints and execution paths related to state changes have all been described in TESG. TESG constructing procedure is also a TOPN model reachability analysis procedure. So if the TESG of one TOPN model has been depicted, the corresponding reachability has also been analyzed.
Similar to the state analysis in TPN, when the TESG of one TOPN model has been completed, the TPN consistency determination theorem can be used to judge the consistency of TOPN models. So the consistency of time critical system can be checked. The theorem can be referenced to Yao's paper (Yao, 1994).

4.2 A modeling and analysis example




```
Var +CT = boolean; /* Transferring Tag */
/*CT is set to "T" in the transition--
"DataFusion" */
Var +Time=Integer; /* Current Relative
Time*/
TT C = with hollow | solid;
TCOT (ComTransf)={
      Fun(CT= =F □ (a≤Time≤b) ):
        ComTransf () □ CT=F □ Mark(p1,C);
      };
/* Mark(P,C): Mark the place P with C。 */
TABT (StateCol)={
      Fun(CT= = F □ (a≤Time≤b)):
        OwnStateCol()□ M(p5,C);
  };
/* M(P,C): Mark the place P with C。 */
Mark(Place,C)={
   Fun(Place is a TABP □ (αp≤Time≤βp)):
      OIP(Place) □ M(Place,C);
   Fun(Place is not in N.TABP): M(Place,C);
};/*mark different places*/
```

Fig. 3. The TOPN Model

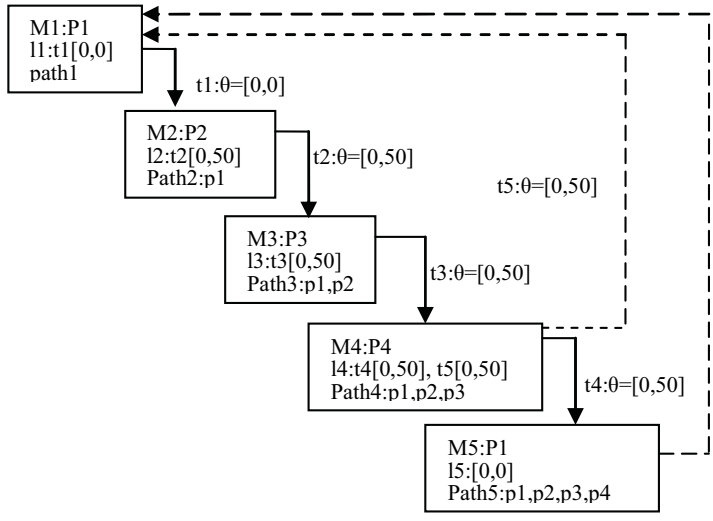


Fig. 4. The TESG of the Decision Model

In distributed cooperative multiple robot systems (CMRS), every robot makes control and schedule decisions according to different system information such as other robot states, its own states and task assignment. The decision making procedure can be divided into 3 main phases. In the first phase, the decision making module collects the above information. For the information mentioned above, every kind of information may include different detailed information. For example, velocity, movement direction and location need to be considered in its and other robot's states. The task to be completed in the future is considered in the task assignment. As the information may not be available from all sensors or sources at the same time moment, the temporal constraint about the information collection is needed. This collection procedure should be completed in 50 unit time. In the second phase, information fusion based method is used to make control and schedule decisions of every robot. To complete the information fusion aim, every kind of information is required simultaneously. It may last for about 50 unit time. Finally, the decision results are transformed to other system modules. The transferring procedure will last for about 50 unit times. In this control procedure, the decision conditions and temporal constraints need to be considered simultaneously, so TOPN is chosen to model this decision making module. Fig.3 has shown the TOPN model of CMRS decision model and its data dictionary respectively. Then Fig.4 has given the state analysis by means of TESG. From the TESG, the design logical errors can be excluded. According to the Yao's consistency judging theorem and the TESG, the TOPN model in Fig.3 is consistent.

5. Fuzzy timed object-oriented Petri net

Although Petri nets can be used to model and analyze different systems, they fail to model the timing effects in dynamic systems. Fuzzy timed Petri net (FTPN) (Pedrycz & Camargo, 2003) has been presented and it has solved this modeling problem, which is on the base of temporal fuzzy sets and Petri nets. However, similar to the general Petri Nets, FTPN may also meet with the complexity problem, when it is used to model complex dynamic systems. In this section, fuzzy timed object-oriented Petri net (FTOPN) is proposed on the base of

TOPN and FTPN, whose aim is to solve the timing effects and other modeling problems of dynamic systems.

5.1 Basic Concept

Similar to FTPN (Pedrycz & Camargo, 2003), fuzzy set concepts are introduced into TOPN (Xu & Jia, 2005-2) (Xu & Jia, 2006). Then FTOPN is proposed, which can describe fuzzy timing effect in dynamic systems.

Definition 13: FTOPN is a six-tuple, FTOPN= (OIP, ION, DD, SI, R, I) where

1. Suppose OIP=(oip, pid, M_0, status), where oip, pid, M_0 and status are the same as those in HOONet (Hong & Bae, 2000) and TOPN (Xu & Jia, 2006).
    - oip is a variable for the unique name of a FTOPN.
    - pid is a unique process identifier to distinguish multiple instances of a class, which contains return address.
    - M_0 is the function that gives initial token distributions of this specific value to OIP.
    - status is a flag variable to specify the state of OIP.
2. ION is the internal net structure of FTOPN to be defined in the following. It is a variant CPN that describes the changes in the values of attributes and the behaviors of methods in FTOPN.
3. DD formally defines the variables, token types and functions (methods) just like those in HOONet (Hong & Bae, 2000) and TOPN (Xu & Jia, 2006).
4. SI is a static time interval binding function, SI: {OIP}→Q*, where Q* is a set of time intervals.
5. R: {OIP} → r, where r is a specific threshold.
6. I is a function of the time v. It evaluates the resulting degree of the abstract object firing.

Definition 13: An internal object net structure of TOPN, ION = (P,T,A,K,N,G,E,F,M_0)

1. P and T are finite sets of places and transitions with time restricting conditions attached respectively.
2. A is a finite set of arcs such that P∩T=P∩A=T∩A=Φ.
3. K is a function mapping from P to a set of token types declared in DD.
4. N, G, and E mean the functions of nodes, guards and arc expressions, respectively. The results of these functions are the additional conditions to restrict the firing of transitions. So they are also called additional restricting conditions.
5. F is a special arc from any transitions to OIP, and notated as a body frame of ION.
6. M_0 is a function giving an initial marking to any place the same as those in HOONet (Hong & Bae, 2000) and TOPN (Xu & Jia, 2006). □

Definition 14: A set of places in TOPN is defined as P=PIP∪TABP, where

1. Primary place PIP is a three-tuple: PIP =(P,R,I), where
    - P is the set of common places similar to those in PN (Murata, 1989) (Peterson, 1991).
2. Timed abstract place (TABP) is a six-tuple: TABP= TABP(p_n, refine state, action, SI, R, I), where
    - p_n is the identifier of the abstract timed place.

- refine state is a flag variable denoting whether this abstract place has been refined or not.
- action is the static reaction imitating the internal behavior of this abstract place.

3. SI, R and I are the same as those in *Definition 1*.□

Definition 15: A set of transitions in TOPN can be defined as T= TPIT∪TABT∪TCOT, where

1. Timed primitive transition TPIT = TPIT (BAT, SI), where
   - BAT is the set of common transitions.
2. Timed abstract transition TABT= TABT (tn, refine state, action, SI), where
   - tn is the name of this TABT.
3. Timed communication transition TCOT=TCOT (tn, target, comm type, action, SI).
   - tn is the name of TCOT.
   - target is a flag variable denoting whether the behavior of this TCOT has been modeled or not. If target = "Yes", it has been modeled. Otherwise, if target = "No", it has not been modeled yet.
   - comm type is a flag variable denoting the communication type. If comm type ="SYNC", then the communication transition is a synchronous one. Otherwise, if comm type="ASYN", it is an asynchronous communication transition.
4. *SI is* the same as that in *Definition 1*.
5. *refine state* and *action* are the same as those in *Definition 3*. □

Similar to those in FTPN (Pedrycz & Camargo, 2003), the object t fires if the foregoing objects come with a nonzero marking of the tokens; the level of firing is inherently continuous. The level of firing ($z(v)$) assuming values in the unit interval is governed by the following expression:

$$z(v) = (\underset{i=1}{\overset{n}{T}}(r_i \rightarrow x_i(v'))\, sw_i)\, t\, \mathrm{I}(v) \tag{1}$$

where T (or t) denotes a t-norm while "s" stands for any s-norm. "v" is the time instant immediately following v'. More specifically, $x_i(v)$ denotes a level of marking of the i^{th} place. The weight w_i is used to quantify an input coming from the i^{th} place. The threshold r_i expresses an extent to which the corresponding place's marking contributes to the firing of the transition. The implication operator (→) expresses a requirement that a transition fires if the level of tokens exceeds a specific threshold (quantified here by r_i).

Once the transition has been fired, the input places involved in this firing modify their markings that is governed by the expression

$$x_i(v)=x_i(v')t(1-z(v)) \tag{2}$$

(Note that the reduction in the level of marking depends upon the intensity of the firing of the corresponding transition, $z(v)$.) Owing to the t-norm being used in the above expression, the marking of the input place gets lowered. The output place increases its level of tokens following the expression:

$$y(v)=y(v')sz(v) \tag{3}$$

The s-norm is used to aggregate the level of firing of the transition with the actual level of tokens at this output place. This way of aggregation makes the marking of the output place increase.
The FTOPN model directly generalizes the Boolean case of TOPN and OPN. In other words, if $x_i(v)$ and w_i assume values in {0, 1} then the rules governing the behavior of the net are the same as those encountered in TOPN.

5.2 Learning in FTOPN

The parameters of FTOPN are always given beforehand. In general, however, these parameters may not be available and need to be estimated just like those in FTPN(Pedrycz & Camargo, 2003). The estimation is conducted on the base of some experimental data concerning marking of input and output places. The marking of the places is provided as a discrete time series. More specifically we consider that the marking of the output place(s) is treated as a collection of target values to be followed during the training process. As a matter of fact, the learning is carried in a supervised mode returning to these target data.
The connections of the FTOPN (namely weights w_i and thresholds r_i) as well as the time decay factors α_i are optimized (or trained) so that a given performance index Q becomes minimized. The training data set consists of (a) initial marking of the input places $x_i(0),\ldots,x_n(0)$ and (b) target values—markings of the output place that are given in a sequence of discrete time moments, that is target(0), target(1),…, target(K).
In our FTOPN, the performance index Q under discussion assumes the form of the following sum:

$$Q=\sum_{k=1}^{K}(target(k)-y(k))^2 \tag{4}$$

where the summation is taken over all time instants (k =1, 2,… , K).
The crux of the training in FTOPN models follows the general update formula being applied to the parameters:

$$\text{param(iter+1)=param(iter)-}\gamma\nabla_{\text{param}}Q \tag{5}$$

where γ is a learning rate and $\nabla_{\mathbf{param}}Q$ denotes a gradient of the performance index taken with respect to all parameters of the net (here we use a notation **param** to embrace all parameters in FTOPN to be trained).
In the training of FTOPN models, marking of the input places is updated according to the following form:

$$\tilde{x_i} = x_i(0)T_i(k) \tag{6}$$

where $T_i(k)$ is the temporal decay. And $T_i(k)$ complies with the following form. In what follows, the temporal decay is modeled by an exponential function,

$$T_i(k)=\begin{cases}\exp(-\alpha_i(k-k_i)) & if \quad k>k_i,\\ 0 & others\end{cases} \tag{7}$$

The level of firing of the place can be computed as the following:

$$z = (\overset{n}{\underset{i=1}{T}} \quad ((r_i \to \tilde{x_i}) sw_i)) \tag{8}$$

The successive level of tokens at the output places and input places can be calculated as:

$$y(k) = y(k\text{-}1)sz,\ x_i(k) = x_i(k\text{-}1)t(1\text{-}z) \tag{9}$$

We assume that the initial marking of the output place y(0) is equal to zero, y(0)=0. The derivatives of the weights w_i are computed as follows:

$$\frac{\partial}{\partial w_i}(t\arg et(k) - y(k))^2 = -2(t\arg et(k) - y(k)\frac{\partial y(k)}{\partial w_i}) \tag{10}$$

where i=1,2,…, n. Note that y(k+1)=y(k)sz(k).

5.3 A modeling example

In cooperative multiple robot systems (CMRS), every robot is controlled according to different system information such as other robot states, its own states and task assignment. As the information may not be available from all sensors or sources at the same time moment, the one that occurs earlier needs to be discounted over time as becoming less relevant. That is to say, information timing effects exist in this kind of dynamic systems. However, in the control of every robot system, every kind of information is required simultaneously. As the information readings could come at different time instants and be collected at different sampling frequency, we encounter an inevitable timing effect of information collected by the system and sensors. It becomes apparent that its relevance is the highest at the time moment when the system sensor captures it but then its relevance has to be discounted over the passage of time. This is an effect of aging that has to be viewed as an integral part of the model. So FTOPN is used to model our CMRS. At the same time, FTOPN can reduce the model complexity and can model complex decision making processes in different levels, because of the OO abstraction concept supported in FTOPN. It triggers interest in the class of the FTOPN.

5.3.1 CMRS example

In our experiment, there are two cooperative robots. FTOPN is used to model the information fusion process in the decision making of scheduling robot in every robot. Because the model is hierarchical, only the highest level of the model is depicted in Fig.5.

In the model of Fig.5, 3 place objects are used to represent 3 kinds of information to be fused. Each kind of information may include different detailed contents. For example, "other robot state" may include other robots' working state, location, speed, movement direction, etc al. So every kind of information is also an abstract object. On the other hand, the relative firing temporal interval is [a, b] of the object. The information should be sampled and processed in this relative interval. So does command sending. If the relative time exceeds it, the information should be sampled again and task should be reassigned. In the model, one transaction object represents the information fusion process. The timing

effect on the fusion is depicted in Fig.6. The information "other robot state" and "own state" complies with the rule in Fig.6 (1). The other information complies with Fig.6 (2). After the fusion, a new command will be sent in this relative interval. The command to be sent is also a place object, which includes robot schedule and control commands.

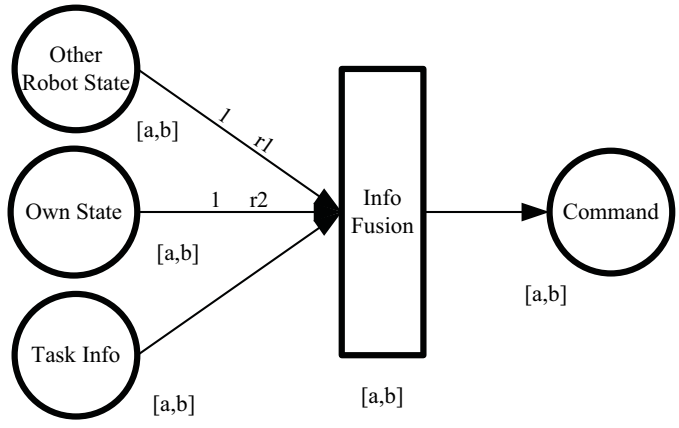


Fig. 5. The FTOPN Model

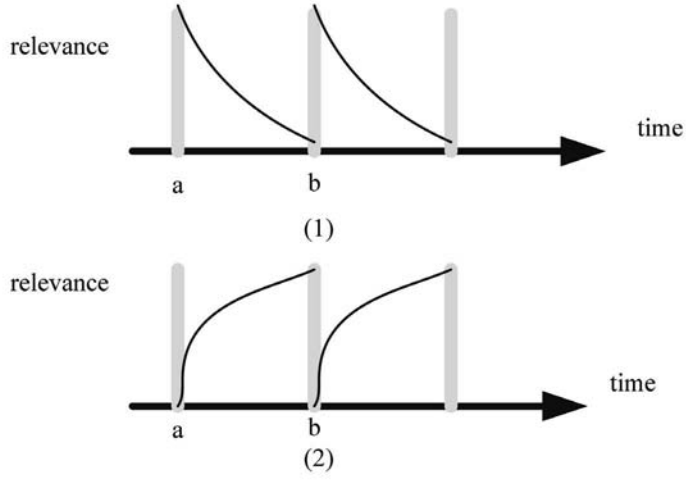


Fig. 6. The Relevance

What's more, all the objects in Fig.5 can also be depicted in details by FTOPN. For example, the object—"Other Robot State" in Fig.5 can also be modeled concretely with FTOPN. The detailed model of the object is depicted in Fig.7. It is also an independent fuzzy reduction process. According to the modeling and analysis requirements, the detailed model can be unfolded directly in the model of Fig.5. At the same time, its training can be conducted independently. It can also be reduced independently and the reduction results will be used

as the believing effect of the corresponding object in the higher level of the FTOPN model in Fig.5.
After completing the FTOPN model, the learning algorithm of FTOPN can be used to train the model and adjust it to fulfill the practical requirements.

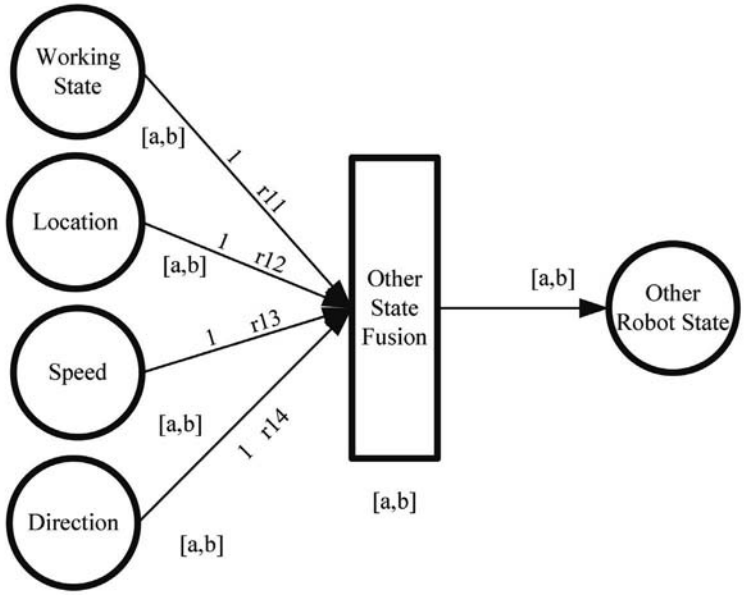


Fig. 7. The Object-"Other Robot State" Model

5.3.2 Application analysis

From the view of the former FTOPN modeling example, objects in FTOPN model can be abstracted. They can be modeled and represented in other levels independently. At the same time, the training and fuzzy reduction can also be conducted independently. So for the abstraction concepts supported, the model complexity has been reduced effectively because of the abstraction concepts in FTOPN. And the fuzzy reduction procedures have been simplified. Essentially, hierarchical modeling idea in FTOPN is to the control model size by abstracting objects in FTOPN model. In nature, OO abstraction concepts are used to control fuzzy knowledge granularity in FTOPN. Because OO concepts are supported in FTOPN, the abstract objects can be unfolded or abstracted in FTOPN model flexibly. Our modeling focus can also be paid upon the important parts.
A comparative analysis between FPN, PN and neural network is conducted in (Pedrycz, 1999). Table.1 summarizes the main features of the fuzzy timed Object-oriented Petri nets and contrasts these with the structures with which the proposed constructs have a lot in common, namely FPN and TFPN. It becomes apparent that FTOPN combine the advantages of both FPN in terms of their learning abilities and the glass-style of processing (and architectures) of Petri nets with the abstraction of OO concepts.

Characteristics	Object Petri nets	Fuzzy Petri Nets	Fuzzy Timed Object Oriented Petri nets
Learning Aspects	From non-existent to significantly limited (the same as those of common Petri nets).	Significant learning abilities parametric optimization of the connections of the net. Structural optimization can be exercised through a variable number of the transitions utilized in the network.	Significant learning abilities as well as FPN. Distributed learning (training) abilities are supported in different independent objects on various system model levels.
Knowledge Representation Aspects	Glass Box or black box style knowledge representation supporting as a result of abstracting a given problem (problem specification) onto the structure of the net in different levels. Well-defined semantics of places and transitions	Transparent knowledge representation (glass box processing style) the problem (its specification) is mapped directly onto the topology of the fuzzy Petri net. Additionally, fuzzy sets deliver an essential feature of continuity required to cope with continuous phenomena encountered in a vast array of problems (including classification tasks)	Glass Box Style (Transparent Knowledge Representation) and Black Box Processing are supported at the same time. The problem (its specification) is mapped directly onto the topology of FTOPN. Knowledge representation granularity reconfiguration reacts on the reduction of model size and complexity.

Table.1 Object Petri nets, Fuzzy Petri nets and Fuzzy Time Object-oriented Petri nets: a comparative analysis

6. Fuzzy timed agent based Petri net

As a typical multi-agent system (MAS) in distributed artificial intelligence (Jennings et al., 1998), when CMRS is modeled, some difficulties are met with. For modeling this kind of MAS, object-oriented methodology has been tried and some typical agent objects have been proposed, such as *active object*, etc (Guessoum & Briot, 1999). However, agent based object models still can not depict its structure and dynamic aspects, such as cooperation, learning, temporal constraints, etc(Jennings et al., 1998). This section proposes a high level PN called fuzzy timed agent based Petri net (FTAPN) on the base of FTOPN (Xu & Jia, 2005-1)

6.1 Agent object and FTAPN

The *active object* concept (Guessoum & Briot, 1999) has been proposed to describe a set of entities that cooperate and communicate through message passing. To facilitate implementing *active object* systems, several frameworks have been proposed. ACTALK is one of the typical examples. ACTALK is a framework for implementing and computing various *active object* models into one object-oriented language realization. ACTALK implements asynchronism, a basic principle of *active object* languages, by queuing the received messages into a mailbox, thus dissociating message reception from interpretation. In ACTALK, an *active object* is composed of three component classes: *address*, *activity* and *activeObject* (Guessoum & Briot, 1999).

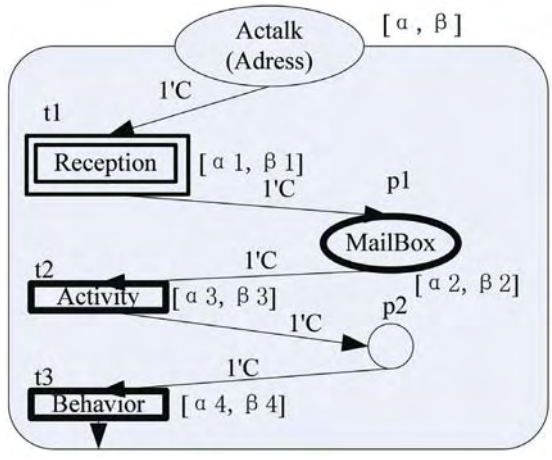


Fig. 8. The FTOPN Model of ACTALK

ACTALK model is the base of constructing *active object* models. However, *active object* model is the base of constructing multi-agent system model or agent system model. So, as the modeling basis, ACTALK has been extended to different kinds of high-level agent models. Because of this, ACTALK is modeled in Fig.8 by FTOPN.

In Fig.8, OIP is the describer of the ACTALK model and also represents as the communication address. One communication transition is used to represent as the behavior of message reception. According to the communication requirements, it may be synchronous or asynchronous. If the message has been received, it will be stored in the corresponding mail box, which is one first in and first out queue. If the message has been received, the next transition will be enabled immediately. So mail box is modeled as abstract place object in FTAPN. If there are messages in the mail box, the following transition will be enabled and fired. After the following responding *activity* completes, some *active behavior* will be conducted according to the message.

Fig.8 has described the ACTALK model based on FTOPN on the macroscopical level. The detailed definition or realization of the object "*Activity*" and "*Behavior*" can be defined by FTOPN in its parent objects in the lower level. The FTOPN model of ACTALK can be used as the basic agent object to model agent based systems. That is to say, if the agent based model—ACTALK model is used in the usual FTOPN modeling procedure, FTOPN has been

extended to *agent based modeling methodology*. So it is called *fuzzy timed agent based Petri net* (*FTAPN*).

6.2 Learning in FTAPN

The parameters of FTAPN are always given beforehand. In general, however, these parameters may not be available and need to be estimated just like those in FTPN (Pedrycz & Camargo, 2003). The estimation is conducted on the base of some experimental data concerning marking of input and output places. The marking of the places is provided as a discrete time series. More specifically we consider that the marking of the output place(s) is treated as a collection of target values to be followed during the training process. As a matter of fact, the learning is carried out in a supervised mode returning to these ***target*** data.

The connections of the FTOPN (namely weights w_i and thresholds r_i) as well as the time decay factors α_i are optimized (or trained) so that a given performance index Q becomes minimized. The training data set consists of (a) initial marking of the input places $x_i(0),\ldots,x_n(0)$ and (b) target values—markings of the output place that are given in a sequence of discrete time moments, that is target(0), target(1),…, target(K).

In FTAPN, the performance index Q under discussion assumes the following form.

$$Q=\sum_{k=1}^{K}(t\arg et(k)-y(k))^2 \tag{11}$$

where the summation is taken over all time instants (k =1, 2,… , K).

The crux of the training in FTOPN models follows the general update formula in the following equation being applied to the parameters:

$$\mathbf{param}(\text{iter}+1)=\mathbf{param}(\text{iter})-\gamma\nabla_{\mathbf{param}}Q \tag{12}$$

where γ is a learning rate and $\nabla_{\mathbf{param}}Q$ denotes a gradient of the performance index taken with respect to all parameters of the net (here we use a notation **param** to embrace all parameters in FTOPN to be trained).

In the training of FTOPN models, marking of the input places is updated according to the following equation:

$$\tilde{x_i} = x_i(0)T_i(k) \tag{13}$$

where $T_i(k)$ is the temporal decay. And $T_i(k)$ complies with the form in the following equation. In what follows, the temporal decay is modeled by an exponential function,

$$T_i(k)=\begin{cases}\exp(-\alpha_i(k-k_i)) & if \quad k>k_i,\\ 0 & others\end{cases} \tag{14}$$

The level of firing of the place can be computed as the following equation:

$$z=(\overset{n}{\underset{i=1}{T}}\ ((r_i\rightarrow\tilde{x_i})sw_i)) \tag{15}$$

The successive level of tokens at the output place and input places can be calculated as that in the following equation:

$$y(k) = y(k-1)sz,\ x_i(k) = x_i(k-1)t(1-z) \tag{16}$$

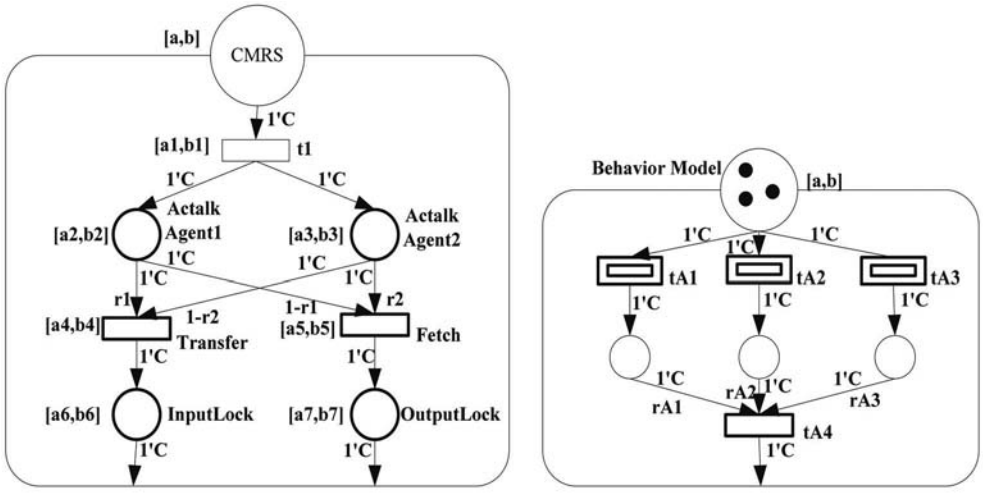
We assume that the initial marking of the output place y(0) is equal to zero, y(0)=0. The derivatives of the weights w_i are computed as the form in the following equation:

$$\frac{\partial}{\partial w_i}(t\arg et(k) - y(k))^2 = -2(t\arg et(k) - y(k)\frac{\partial y(k)}{\partial w_i}) \tag{17}$$

where i=1,2,…, n. Note that y(k+1)=y(k)sz(k).

6.3 A Modeling example

In manufacturing integrated circuits, usually there is a Brooks Marathon Express (MX) CMRS platform made up of two transferring robots. These two cooperative robots are up to complete transferring one unprocessed wafer from the input lock to the chamber and fetch the processed wafer to the output lock. Any robot can be used to complete the transferring task at any time. If one robot is up to transfer one new wafer, the other will conduct the other fetching task. They will not conflict with each other. Fig. 9 depicts this CMRS FTAPN model, where two agent objects (ACTALK) is used to represent these two cooperative robots.



(a) The Agent Based FTAPN Model (b) The Behavior Model in Every Agent

Fig. 9. The FTAPN Model

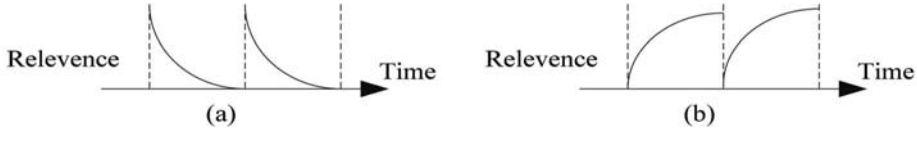


Fig. 10. The Relevance

Fig. 9 (a) has depicted the whole FTAPN model. The agent object—"ACTALK" is used to represent every robot model. Different thresholds are used to represent the firing level of the behavior conducted by the corresponding robot (agent). They also satisfy the unitary requirements and change according to the fuzzy decision in the behavior of every agent in Fig. 9 (b). In the model of Fig. 9 (b), three communication transition objects are used to represent the behavior for getting different kinds of system states. These states include the state of the other robot, its own goal and its current state, which can be required by the conductions of the communication transitions t_{A1}, t_{A2} and t_{A3}. When one condition has been got, the following place will be marked. In order to make control decisions (transition object t_{A4}) in time, all of these state parameters are required in the prescriptive time interval. However, the parameters of the arrival times comply with the rule in Fig. 10 (a). The other two kinds of information comply with that in Fig. 10 (b). After the decision, a new decision command with the conduction probability will be sent in this relative interval and it also affects which behavior will be conducted by updating the threshold in Fig. 10 (a).

6.4 Application aspects of FTAPN

Owing to the nature of the facet of temporal knowledge, fuzzy sets and object-oriented concepts in this extension of PN, they become viable models in a wide range of engineering problem augmenting the already existing high level Petri nets, cf. (Hong & Bae, 2000) (Wang, 1998). Two main and commonly categories of models are worth elaborating here.

6.4.1 Models of multi-agent systems

The multi-agent paradigm and subsequently a variety of models are omnipresent in a number of areas. In a nutshell, in spite of the existing variety of improved models, it still lacks a powerful modeling method, which can bridge the gap between model and practical implementations. Petri nets come with objects, temporal knowledge and fuzzy sets can use active objects to model generic agents with situatedness, autonomy and flexible. This helps us to use the object to reduce the complexity of MAS systems and the dynamic learning and decision to support the autonomy of agents.

6.4.2 Models of complex real-time systems

In models of complex real-time systems as usually encountered in industrial practice, the scale of the system module may be too complicated to be analyzed and the readings of different system state or sensors may be available at different time. The former may lead to the state explosion, while the latter needs adjustment of relevance of the information gathered at different time scales. The object models with temporal information degradation (aging) help to abstract complicated model and quantify the confidence of the inferred results.

7. Conclusion

Firstly, a high-level Petri net called timed hierarchical object-oriented Petri net (TOPN) is studied deeply in this chapter.

For modeling complex time critical systems and analyzing states, TOPN is proposed firstly. The work is based on the following work: Hong's hierarchical object-oriented Petri net (HOONet) (Hong & Bae, 2000), Marlin's timed Petri net (Merlin & Farber, 1976)) and Yao's extended state graph (Yao, 1994).With the introduction of temporal knowledge in TOPN, the temporal constraints need to be considered in state analysis. A state analysis method—"TOPN extended state graph (TESG)" for TOPN has also been presented in this chapter. Not only state analysis, but also consistency can be analyzed by means of TESG. On the other hand, TOPN can model complex time critical systems hierarchically. So analysis of properties and state change becomes much easier. A decision making example modeled by TOPN has been used to illustrate the usefulness of TOPN.

In the future research of TOPN, temporal reasoning and TOPN reduction rules will be studied, which can be used to refine and abstract TOPN models with preserving timing property.

Secondly, fuzzy timed object-oriented Petri net (FTOPN) is presented on the base of TOPN.

Timing effect is also a usual phenomenon in dynamic systems especially in time critical systems. In order to model, analyze and simulate this kind of systems, this paper proposes fuzzy timed object-oriented Petri net (FTOPN) on the base of TOPN (Xu & Jia, 2006) and FTPN (Pedrycz & Camargo, 2003). Temporal fuzzy sets are used in FTOPN to describe the timing effect and evaluation levels can be got according to the information arriving time and specific fuzzy relevance function. What's more, compared with FTPN (or FPN) models, the model size and reduction complexity of FTOPN models can be reduced by controlling object granularity because of supporting OO concept in FTOPN. Every abstract object in FTOPN can be trained and reduced independently according to the modeling and analysis requirements for OO concepts supported in FTOPN. The validity of this modeling method has been demonstrated by using it in the simulation of the decision information fusion process in our CMRS.

State analysis which can analyze the FTOPN and FTAPN models, needs to be studied in the future research. With the temporal fuzzy sets introduced into TOPN, the certainty factor about object firing (state changing) needs to be considered in the state analysis.

Finally, agent concepts are introduced into FTOPN and fuzzy timed agent based Petri net (FTAPN) is proposed in this chapter.

Cooperative multi robot system is a kind of usual manufacturing equipments in manufacturing industries. In order to model, analyze and simulate this kind of systems, this paper proposes fuzzy timed agent based Petri net (FTAPN) on the base of FTOPN (Xu & Jia, 2005-1) and FTPN (Pedrycz & Camargo, 2003). In FTAPN, one of the *active objects*—ACTALK is introduced and used as the basic agent object to model CMRS, which is a typical multi-agent system. Every abstract object in FTOPN can be trained and reduced independently according to the modeling and analysis requirements for OO concepts supported in FTOPN. The validity of this modeling method has been used to model Brooks CMRS platform in manufacturing IC. The FTAPN can not only model complex MAS, but also be refined into the object-oriented implementation easily. It has provided a methodology to overcome the development problems in agent-oriented software engineering. At the same time, it can also be regarded as a conceptual and practical artificial

intelligence (AI) tool for integrating MAS into the mainstream practice of software development.

State analysis needs to be studied in the future. An extended State Graph (Xu & Jia, 2006) has been proposed to analyze the state change of TOPN models. With the temporal fuzzy sets introduced into FTAPN, the certainty factor about object firing (state changing) needs to be considered in the state analysis.

8. Acknowledgement

This work is jointly supported by the National Nature Science Foundation (Grant No: 60405011, 60575057) and the China Postdoctoral Foundation for China Postdoctoral Science Fund (Grant No: 20040350078) in China.

9. References

Bastide, R.(1995). *Approaches in unifying Petri nets and the object-oriented approach*, Proceedings of the 1st International Workshop on Object-oriented Programming and Models of Concurrency, http://wrcm.dsi.unimi.it/PetriLab/ws95/home.html, Turin, Italy, June, 1995

Battiston, E., Cindio, F.D., Mauri, G.(1988). *OBJSA Nets: a class of high-level nets having objects as domains,* Proceedings of APN'88, Lecture Notes in Computer Science, Vol.340, pp.20-43

Biberstein, O., Buchs, D.(1994). *An object-oriented specification language based on hierarchical algebraic Petri nets*, Proceedings of the IS-CORE Workshop, Amsterdam, Holland, September 1994

Bucci, G., Vivario, E.(1995). *Compositional validation of time-critical systems using communicating time Petri nets,* IEEE Transactions on Software Engineering, Vol.21, No.12, pp.969-992

Florin, G., Fraize, C., Natkin, S.(1991). *Stochastic Petri nets: Properties, applications and tools*, Microelectron. Reliab., Vol. 31, No. 4, pp. 669–697

Guessoum, Z., Briot, J.P.(1999). *From active objects to autonomous agents,* IEEE Concurrency, Vol.7, No.3, pp. 68 – 76

Harel, D., Gery, E.(1996). *Executable object modeling with statechart,* Proceedings of the 18th International Conference on Software Engineering, Germany, March 1996, pp. 246±257.

Hong, J.E., Bae, D.H.(2000). *Software Modeling And Analysis Using a Hierarchical Object-oriented Petri net,* Information Sciences, Vol.130, pp.133-164

Jennings, N.R., Sycara, K., Wooldridge, M.(1998). *A Roadmap of Agent Research and Development,* Autonomous Agents and Multi-Agent Systems, Vol.1, pp.7–38

Jensen, K.(1992). *Coloured Petri Nets: Basic Concepts, Analysis methods and Practical Use,* Springer, ISBN : 3-540-60943-1, Berlin, German

Lakos, C., Keen, C.(1994). *LOOPN++: a new language for object-oriented Perti nets,* Technical Report R94-4, Networking Research Group, Univesity of Tasmania, Australia, April 1994

Merlin, P., Farber, D.(1976). *Recoverability of communication protocols – Implication of a theoretical study* , IEEE Transactions on Communications, Vol.COM-24, pp.1036-1043

Moody, J.O., Antsaklis, P.J. (1998). *Supervisory Control of Discrete Event Systems Using Petri Nets,* Kluwer Academic Publishers, ISBN-10: 0792381998, MA, USA

Murata, T.(1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of IEEE,* Vol.77, No.4, (April 1989) pp.541-580

Pedrycz, W.(1999). *Generalized fuzzy Petri nets as pattern classifiers,* Pattern Recognition Letters, Vol.20 , pp.1489-1498

Pedrycz, W., Camargo, H.(2003). *Fuzzy timed Petri nets*, Fuzzy Sets and Systems, Vol.140, No. 2, pp. 301-330

Peterson, J.L.(1991). *Petri Net Theory and the Modeling of Systems,* Prentice-Hall, ISBN:0-13-661983-5, N.J., USA

Ramchandani, C.(1974). *Analysis of Asynchronous Concurrent Systems by Timed Petri nets,* Massachusetts Institute of Technology, Project MAC, Technology Report 120, 1974

Schuman, S.A.(1997). *Formal Object-oriented Development,* Springer, ISBN-10: 3540199780, Berlin, German.

Sloan, R., Buy, U.(1996). *Reduction Rules for Time Petri Nets*, Acta Inform, Vol.33, pp.687-706

Tsai, J., Yang, S., Chang, Y.(1995). *Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications,* IEEE Transactions On Software Engineering, Vol.21, No.1, pp.32-49

Wang, J.(1998). *Timed Petri Nets – Theory and Application,* Kluwer Academic Publishers, ISBN : 0-7923-8270-6, Boston , USA

Wang, J. ; Deng, Y., Zhou, M.(2000). *Compositional time Petri nets and reduction rules,* IEEE Transactions on Systems, Man and Cybernetics (Part B), Vol. 30, No.4, (Aug. 2000) pp. 562 -572

Xu, H., Jia, P.F.(2005-1). *Fuzzy Timed Object-Oriented Petri Net,* Artificial Intelligence Applications and Innovations (Proceedings of AIAI2005), Springer, pp.148-160, N.Y., USA

Xu, H., Jia, P.F.(2005-2). *Timed Hierarchical Object-oriented Petri Net,* GESTS International Transaction on Computer Science and Engineering, Vol.24, No.1, pp.65-76

Xu, H., Jia, P.F.(2006). *Timed Hierarchical Object-Oriented Petri Net-Part I: Basic Concepts and Reachability Analysis,* Lecture Notes In Artificial Intelligence (Proceedings of RSKT2006), Vol. 4062, pp.727-734

Xu, H., Jia, P.F.(2007). *A Novel Modeling Method for Cooperative Multi-Robot Systems Using Fuzzy Timed Agent Based Petri Nets,* LNCS (Proceedings of ICCS2007), Vol.4488, pp.956-959

Yao, Y.L.(1994). *A Petri Net Model for Temporal Knowledge Representation and Reasoning,* IEEE Transactions On Systems, Man and Cybernetics, Vol.24, pp.1374-1382

Zhou, M.C.(1995). *Petri Nets in Flexible and Agile Automation,* Kluwer Academic Publishers, ISBN : 0792395573 , MA, USA

13

Scheduling Analysis of FMS Using the Unfolding Time Petri Nets

Jong kun Lee[1] and Ouajdi Korbaa[2]
*Changwon National University[1], **Changwon***
Ecole Centrale de Lille[2], Lille
Korea[1]
France[2]

1. Introduction

FMS (flexible manufacturing system) is composed of a set of versatile machines, zig and fixture, and automatic transport system for moving parts between each job. In FMS, one of the important subjects is to formulate the general cyclic state-scheduling problem to minimize the WIP in order to satisfy economical constraints. Various methods have been proposed by researchers (Zuberek, 1987; Korbaa, 1997; Richard, 1998; Murata,1989; Kondratyev, 1996; Hwang, 1997; Liu, 1999; Lee, 2004)) to solve this problem. Hillion (Hillion, 1987) proposed a heuristic algorithm based on the computation of the degree of feasibility after giving a Petri net (PN) model of the system. Also, Valentin (Valentin, 1994) improved this algorithm by using Timed hybrid Petri nets and by introducing available intervals concept. The problem was that this algorithm could not guarantee the feasibility of the solution in one run. Korbaa (Korbaa, 1997) developed an algorithm to find near-optimal solution using the regrouping algorithm. Korbaa tried to get near-optimal solution and to minimize the WIP. For getting an optimal solution long computational time has been required. This means that all these methods have the problem of complexity and computational time. To simplify the calculation process in the scheduling problem and to make shorter computational time for getting many feasible solutions, we consider unfolding Petri nets to analyze the sequence process and to explain the reduced process. We call "unfolding" a PN unfolding, which has the reachability properties of the original net. Structural analysis on "unfolding" is much easier than on the initial model. The advantage of unfolding is that the state space explosion can be avoided since it is based on partial order semantics. To reduce the processing time, we consider an algorithm to select an environment of a shared resource, which has priority over the other ones, using the transitive matrix. In our case, the system has been analyzed to get the best possible solutions based on this environment of shared resource. The model has been divided into slices and create PN slice using this concept. The properties of a PN can be classified into categories: behavioral and structural properties. The behavioral properties are investigated in association with the marking of PN, e.g., reachability, boundness and liveness (Liu, 1999). Both transition and place invariants belong to structural properties in PN. If the behavioral

properties used by the transitive matrix are exhibited, it could be easier to analyze the system after slicing off some subnets.
This paper is organized as follows: some definitions of Time Petri Nets(TPN) and unfolding are given in Section 2, and time Petri net slice is presented in Section 3, The scheduling objectives and outline the problems arising during the transformation of the initial model into an unfolding TPN are described in Section 4. In section 5, we introduce an illustration model for FMS, and specially emphasize the different ways of obtaining a closed loop model. In section 6, we are showing the benchmarking resultants after analyze using the performance evaluation factors. A conclusion and some perspectives will end this paper.

2. Basic notions of time Petri nets and slices

In this section certain terms that are often used in this paper are defined (Best,1990; Carlier, 1988; Camus, 1997; Esparza, 1996; Julia, 1995; Lee, 1995; Lee, 2004; Lee, 2006; Liu,1999; Murata, 19875; Taubin, 1997).
Let N=<P, T, I, O, Mo,τ>be a Time Petri Nets, where P is the set of places (m elements), T is the set of transitions (n elements), and $P \cap T = \phi$, I: T->P^{∞} is the input function, O: T->P^{∞} is the output function. $\mathbf{M_0} \in \mathbf{M} = \{\mathbf{M} \mid \mathbf{M} : \mathbf{P} \rightarrow \mathbf{N}\}$, M_O is an initial marking, **N** is the set of positive integers. τ : T→**N** is a time function which associate to each transition of T a deterministic rational.
A transition $t \in T$ is enabled at a marking $M' = M - {}^{\bullet}t + t^{\bullet}$, where ${}^{\bullet}t$ represents the incoming weights of the fired transition and $t^{\bullet}$ the outgoing weights of the fired transition.
We denote by M (t>M'. The set of reachable marking of N is the smallest set (Mo> containing Mo and such that if M ∈(Mo> and M (t>M' (for some t ∈ T) then M' (t>Mo. A Time Petri Nets N is safe if for every reachable marking M, M(P) ⊆{0,1}; and bounded if there is k ∈ **N** such that M(P) ⊆{0,…,k}, for every reachable marking M. The set of successors of a node x∈ P ∪T is x ={y∈P ∪T;(x,y) ∈F: I ∪O}, and the set of its predecessors is ${}^{\bullet}x$={y ∈P ∪T; (y,x) ∈F: I ∪O}.
A Time Petri nets N_S=<P_S,T_S,F_S,M_S,τ_S>,where $P_S \subseteq$ P, $T_S \subseteq$ T, $F_S \in$F ,M_S ∈(Mo> and $\tau_S \in \tau$ then we can say that N_S is sub-net of N.
The number of occurrences of an input place P_i in a transition t_j is given by #(P_i,I(t_j)), and the number of occurrences of an output place P_i in a transition t_j is given by #(P_i,O(t_j)).
The matrix of the PN structure, S is S=<P,T,C^-,C^+>, where P,T are the finite set of places and transitions, respectively. C^- and C^+ are matrices of m rows (one for each place) by n columns (one for each transition) defined by:

C^- =[I,j] = #(P_i,I(t_j)), matrix of input function,

C^+ =[I,j] = #(P_i,O(t_j)), matrix of output function.

And the incidence matrix C is given by $C=C^+-C^-$.

3. Invariant and transitive matrix

3.1 Invariant

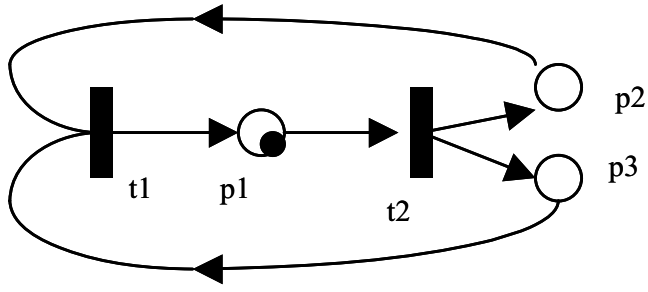
For completeness, we recall the terminologies which were used in (Lee, 2001; Lee,2004; Lee,2006; Liu,1999).

(Def. 3.1): Invariant
A row vector X = $(x_1,x_2,\ldots,x_n)\geq 0$ is called a P-invariant if and only if X •C = 0, where $x\neq 0$ and •denotes the vector product.
A column-vector $Y=(y_1,y_2,\ldots,y_n)^T \geq 0$ is called a T-invariant, if and only if C •Y = 0, where Y is an integer solution of the homogeneous matrix equation and $Y\neq 0$.
(Def.3.2): Place transitive and Transition matrix
The place transitive and transition matrix are defined, respectively as follows:

$$C_P = C^-(C^+)^T$$

$$C_T = (C^+)^T C^-$$

(Example):



$$C_P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Fig. 1. A Petri net

This matrix allows representing the relation between places in terms of output and input. For previous example, we can find that p2 and p3 receive a token after p1 and that p1 receives one token from p2 and another from p3 (Fig.1).
(Def. 3.3): Labeled place transitive matrix

Let L_{CP} be the labeled place transitive matrix:

$$L_{CP} = C^- diag(t_1,t_2,\ldots,t_n)(C^+)^T$$

The elements of L_{CP} describe the direct transferring relation that is from one place to another one through one or more transitions.

(Def. 3.4): Let L_{CP} be the m×m place transitive matrix. If a transition t_k appears s times in the same column of L_{CP}, then we replace t_k in L_{CP} by t_k / s in L^*_{CP}.

Through introducing the m×m place transitive matrix, we can evaluate the transition enabling firing, and calculate the Quantity and the Sequence of Transition enabling Firing.

(Def.3.5): Let a reachable marking $M_R(K+1)$ from $M(k)$ be an m-vector of nonnegative integer. The transformation is defined by:

$$M_R(k+1)^T = M(k)^T L^*_{CP}$$

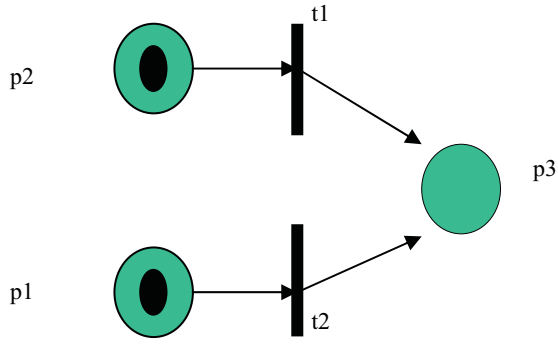


Fig. 2. Illustrative example

(Example):

This example net explained like that $M(k) = [P_1(k), P_2(k), P_3(k)]^T$, then we can obtain (Fig. 2):

$$L_{BP} = \begin{bmatrix} 0 & 0 & t_1 \\ 0 & 0 & t_2 \\ 0 & 0 & 0 \end{bmatrix}$$

and $M_R = [0,0,t_1(P_1(k)) + t_2(P_2(k))]^T$.

3.2 Algorithm and properties

A basic Unit of Concurrency (BUC) is a subnet corresponding to a resource. It contains the incoming transitions of the resource and also the outgoing transitions. We chose the BUC based on the place transitive matrix; in this paragraph we propose an algorithm to determine the BUC.

Algorithm: BUC construction algorithm

Input: N = <P, T, I,O, Mo, τ>

Output: BUC of N, N_S=<P_S, T_S, I_S, O_S,Mo_S, τ_s>

Define L_{CP} and consider one shared resource (machine)

1. Find the all-relational places, of the shared resource, in each column and row L_{CP} and make an element of own BUC with this initial marking place.
2. Find the relational place of selected place in (1).
3. Link all selected places and transitions with existing arcs on the initial Petri net.

The method of partitioning the model divides the system into BUC. The obtained BUC is defined as follows:

(Def.3.6): BUC

In Time Petri net N= <P, T, I,O, Mo, τ>, when the places set is divided by the previous algorithm, the BUCs are defined by (BUC_i | i=1,…,n), and each BUC_i = (P_i,T_i,F_i,M_i,τ_i) satisfies the following conditions.

$$P_i = P\_BUC_i,$$

$$T_i = \{t \in T \mid s \in P_i, (s, t) \in F \text{ or } (t, s) \in F\},$$

$$F_i = \{(p, t) \in F, (t, p) \in F \mid p \in P_i, t \in T_i\},$$

$$\forall \tau_i \in \tau, \tau_i(t) = \tau(t) \text{ and } \forall p \in M_i, M_i(p) = M(p).$$

We can say that the flow of control in the Petri nets is based on the tokens flow. If the token is divided into some tokens after firing a transition, the flow of control divides to several flows. Accordingly, we define that the BUC is a set of the executed flow of control based on the functional properties of the net. In the Time Petri net model, the behavioral condition of transition t are all predecessors' places ($^\bullet t$), which are connected with transition t. Petri net Slices are subnets based on BUC at the transition level, and a functional condition is defined according to the following conditions.

- When transition t is not shared : satisfy the precondition of transition t only.
- When several slices share transition t: satisfy the preconditions in all BUC, which have transition t.

(Theorem. 3.1) Let N be a Time Petri net and Ns be a set of Time Petri net slices that is produced by the slice algorithm. If Ns satisfied the functional conditions, N and Ns are behavioral equivalent (N≡Ns).

(Proof) (We prove this theorem based on the $p \in {}^\bullet t_i \Leftrightarrow p \in {}^\bullet t_T$).

(⇒) Since the p is an element of P_i, $\exists\ t_i \in T_i$, such that $p \in {}^\bullet t_i$. And by the BUC definition, P_i = P_BUC_i, P_BUC_i is a set of place which is obtained by the BUC algorithm, such that $P\_BUC_i \subseteq P$. So, we say that if $p \in {}^\bullet t_i$ is then $p \in {}^\bullet t_T$ is true.

(⇐) Consider if $p \in P$ and $p \notin P\_BUC_i$ then $P \not\subset P\_BUC_i$. If $p \in P$, $t \in T$ such that $p \in {}^\bullet t$. And $p \notin \bigcup_{t \in T_i} {}^\bullet t$, in this case, we can say that $\bigcup_{t \in T_i} {}^\bullet t = \bigcup_{i=1} P_i$ but by the definition of BUC, P_i is a set of places which is obtained by the BUC algorithm, such that $P_i \subseteq P$. So if $p \in P$ and $p \notin P\_BUC_i$ then $P \not\subset P\_BUC_i$ is not true.□

4. Unfolding time Petri nets (UTPN)

Unfolding technique, originally proposed by McMillan(McMillian,1995), is a method used to avoid the state explosion problem in the verification of concurrent systems modeled with finite-state Petri nets. The technique is based on the concept of net unfolding; well-

unknown partial order semantics of Petri nets (Hwang, 1997; Kondratyev, 1996; Lee,2001; McMillian,1995).

4.1 Occurrence Nets

An occurrence net is a net, which represents clearly causal relations between places, especially transitions, of the original net. In this section, we briefly recall the main definitions (Fig. 3 and 4).

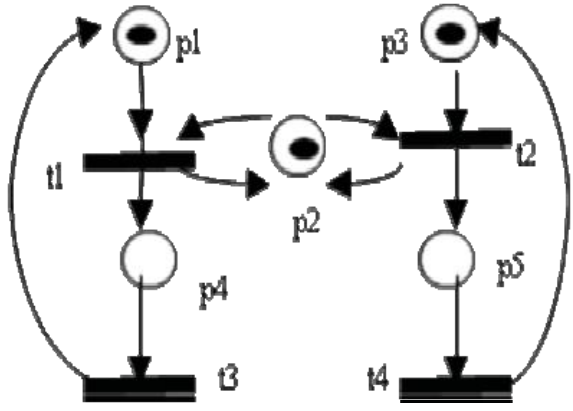


Fig. 3. Cyclic Petri net

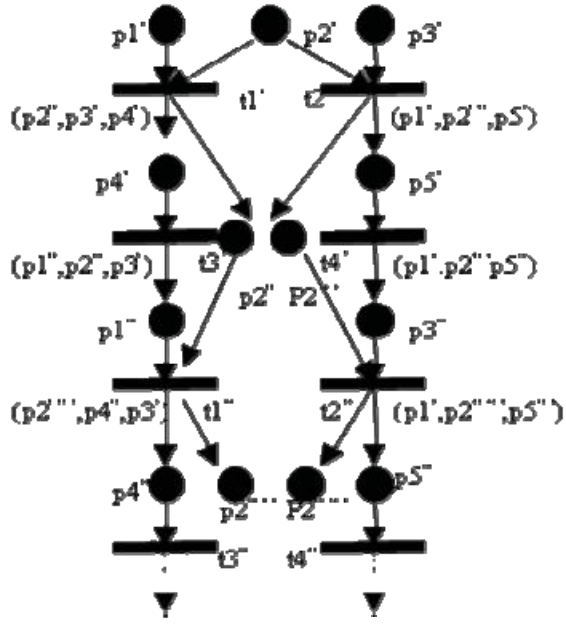


Fig. 4. Occurrence net of Fig. 3.

(Def.4.1) (OCN (Occurrence net)). An occurrence net is an acyclic Time Petri net $N=<P,T,F,M_o,\tau>$ in which every place $p\in P$ has at most one input transition ($|{}^\bullet p|\leq 1$, $F \subseteq (P \times T)\cup (T \times P)$) is the flow relations.
An OCN algorithm is summarized as follows (Hwang, 1997; McMillian,1995):
Algorithm: (Occurrence net construction)
Input: $N=<P,T,F,M_o,\tau>$
Output: the OCN of N, $N'=<P',T',F',M_o',\tau'>$.
1. Make a copy of the place in to the OCN and labeled as p′.

Repeats (2)-(6) until the transitions set becomes empty.
2. Choose a transition $t \in T$
3. For each place in $\bullet t$, find a copy in the OCN, and if not found then go back to (2).

Do not choose same subnet in OCN twice for a given t.
4. If any pair of chosen places is not concurrent, go to (1).
5. Make a copy of t in OCN and labeled as t′. Draw an arc from every places which was chosen to t′.
6. For each place in $t\bullet$, make a copy in the OCN.

Also, we can see a relationship between OCN and cyclic nets as following definition.
(Def.4.2) Let $N' =<P',T',F',M_o',\tau'>$ be an OCN and $N=<P,T,F,M_o,\tau>$ be a cyclic net, and $\exists$ labeling function $L':P\subseteq' P$ and $T' \subseteq T$ then OCN satisfying follows conditions:
1. $\{\exists\ s'|\ s' \subset T', F'^* \subseteq (P' \times T')\cup (T' \times P')\}$
2. $\forall p\in P'$, $p\in t_1^\bullet$ and $p\in t_2^\bullet$ implies t1 =t2.
3. $\forall t_1,t_2,t_3\in T', t_1F'^*t_3$ and $t_2F'^*t_3$ and ${}^\bullet t_1\cap {}^\bullet t_2\neq\varnothing$ implies $t_1 = t_2$.
4. $\forall t_1,t_2\in T'$, $L'(t_1)=L'(t_2)$ and ${}^\bullet t_1={}^\bullet t_2$ implies $t_1 = t_2$.

4.2 Unfolding

Unfolding is used to verify the occurrence net after cut or truncate, based on local configuration and basic marking (Kondratyev, 1996; McMillian,1995).
(Def.4.3)(Configuration). A set of transitions $\mathbf{C}'\subset T'$ is a configuration in an OCN if:
1. for each $t'\in\mathbf{C}'$ the configuration $\mathbf{C}'$ contains t′ together with all its predecessors;
2. $\mathbf{C}'$ contains no transitions in mutual conflict.

(Def.4.4) Let $\mathbf{C}'$ be a configuration of an occurrence net. A final marking of $\mathbf{C}'$, denoted by FM (C′), is a marking reachable from the initial marking after all transitions of C′ and only those transitions are fired. A final marking of a local configuration of t′ is called a basic marking of t′ and denoted BM (t′). The set of predecessor transitions of t′ of the $\mathbf{C}'$ is called the local configuration of t′ and is denoted as $\Rightarrow t'$.
(Def.4.5) (Cut off). A transition t_i' of an occurrence net is a cut off transition, if there exists another transition t_j' such that BM (t_i')=BM (t_j') and $|\Rightarrow t_i'| > |\Rightarrow t_j'|$, where $|\Rightarrow t_i'|$ is the cardinality of $\{\Rightarrow t_i'\}$. An unfolding is the greatest backward closed subnet of an occurrence net containing no nodes after cut-off transition.
(Def.4.6) An unfolding time Petri net $UTPN=<P',T',F',M_o,\tau'>$ is obtained from the occurrence net by removing all the places and transitions, which succeed the cut-off (Fig. 5).

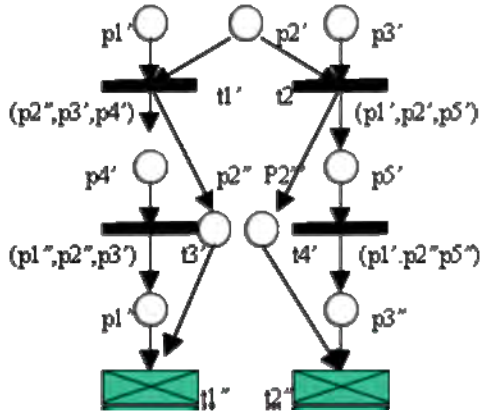


Fig. 5. Unfolding net of Fig. 3.

Let UTPN be a unfolding Time Petri net, Mo be the initial marking and FM be a final marking, find a sequence x such that: BM (x>FM, and the reachability delay is minimal (Richanrd,1998):

$$\text{Min } z = \sum_{t \in T} x_i$$

Subject to BM (x>FM where X=(x_t) is the characteristic vector of the sequence x.

5. Modeling of UTPN

An important sub-class (sliced net) of Time Petri nets is a net in which has an independent control flow and for which all weights associated to arcs are equal to one. In this works, we select an independent control flow based on the machines operations. This Time Petri net can perfectly model the command we want to implement. At the end of the optimization approach we obtained a model where all the operating sequence are linear and the next step is to compute the best schedule.

5.1 Computing the optimal schedule

The schedule of a sliced net can be easily computed by playing the token and firing transition as soon as possible. We can compute firing dates of transitions by computing potentials on the sliced net after unfolding. In the unfolding net UTPN, it has one function for compute the makespan time, as *f* (UTPN). The function *f* (UTPN) is the necessary time to go from the initial marking Mo to the object marking. *f* (UTPN) is composed of $h(t_i)$ and $g(t_i)$, where $h(t_i)$ is an operating time of transition t_i, and $g(t_i)$ is an operating time of the next transition t_i (we call it minimum waiting time to fire transition t_i). Then, the schedule duration can be computed by the following recurrent formula:

$$f(\text{UTPN}) = h(t_i) + g(t_i)$$

and

$$f(\mathrm{UTPN}) = \underset{1\le j\le n}{\mathrm{Max}} \{(\sum_{i=1}^{j} h(t_i)) + g(t_j)\}$$

The best schedule of the UTPN is obtained by minimizing the function f(UTPN). We can say that if we find a minimized unfolding net, the schedule of marking is an optimized schedule. So let UTPN be the makespan time of the optimized unfolding schedule. Then the time can be computed using the unfolding net as follows:

$$\text{Optimize UTPN} = \min (f(\mathrm{UTPN}_i))$$

To apply our method continuously, we consider some notations about degree of operation time in the machine and minimized WIP. The deciding order is one of the important things in the scheduling problem. We consider a throughput of the operation time in the machine, based on the number of tokens (number of resource share), and the operating time of machine.

Let $d(m_i)$ be the degree of operation time in the machine, then

$$d(m_i) = \frac{\varphi(m_i)}{\gamma(m_i)}$$

where, $\varphi(m_i)$ is total operation time of the machine i, and $\gamma(m_i)$ is number of token (resource share transition) in machine i.

So, this degree of $d(m_i)$ is one parameter to choice the select an order to apply in Unfolding net

In the linear cyclic scheduling problem, the minimization of the number of pallets is one of the important factors.

(Def. 5.1) Let CT be an optimal cyclic time based on the machines work, then WIP lower bound is:

$$\mathrm{WIP} = \left\lceil \frac{\sum_{i} \left(\sum_{\mathrm{OStobecarriedbyi}} \mathrm{Operatingtime} \right)}{\mathrm{CT}} \right\rceil$$

5.2 Dispatching rules

We consider a system with two machines such as M1, M2 and two jobs such as OP1 and OP2 in (Camus, 1996). Let us assume that the production ratio corresponds to the production of 50% of OP1, 50% of OP2(Fig. 6).

Now, we show the transitive matrix of the example as shown in Table 1. For simplicity reason, we ignore two places w1, w2 and one transition tw. In this table, initial tokens exist in M1 and M2. The cycle time of OP1 and Op2 are 11 and 9, and the working time of machine M1 and M2 are both 10, respectively. So the cycle time CT is 10. The minimized WIP is:

$$\mathrm{WIP} = \left\lceil \frac{11+9}{10} \right\rceil = 2$$

Also, about the degree of the feasibility time of the machine, M1 and M2 have same priority,

$$d(M1) = \frac{10}{3} = 3.3$$

$$d(M2) = \frac{10}{3} = 3.3$$

These machines have same operating time (i.e. 10) for three operations showing that it is not important to give the priority for first approach. So we select M1 to start with.

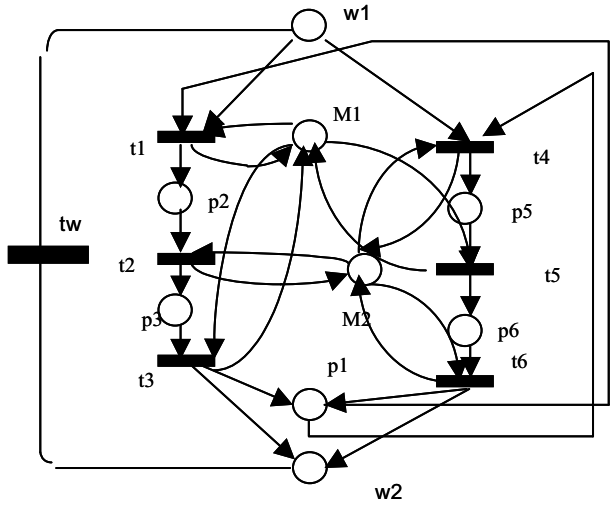


Fig. 6. A two shared resources example

$$L_{CP} = \begin{bmatrix} 0 & t1/2 & 0 & t4/2 & 0 & t1/2 & t4/2 \\ 0 & 0 & t2/2 & 0 & 0 & 0 & t2/2 \\ t3/2 & 0 & 0 & 0 & 0 & t3/2 & 0 \\ 0 & 0 & 0 & 0 & t5/2 & t5/2 & 0 \\ t6/2 & 0 & 0 & 0 & 0 & 0 & t6/2 \\ & & & & & t1/2 & \\ t3/2 & t1/2 & 0 & 0 & t5/2 & t3/2 & 0 \\ & & & & & t5/2 & \\ & & & & & & t6/2 \\ t6/2 & 0 & t2/2 & t4/2 & 0 & 0 & t2/2 \\ & & & & & & t4/2 \end{bmatrix}$$

Table 1. Transitive matrix of the illustration example

Now, we select row and column of p1, p3, p5, and M1 in (Table 1), and make one slice net based on the selected places and transitions.

1. Slice BUC of M1 and its unfolding nets

Machine M1 involved two tasks (OP1and OP2) in three processes (t1, t3, t5)(Fig. 7 and 8).

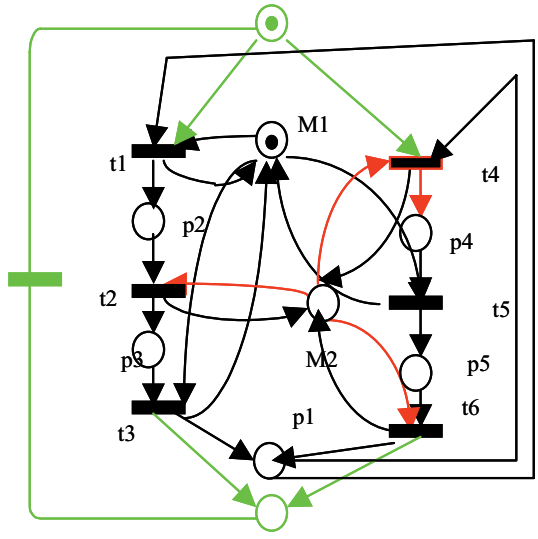
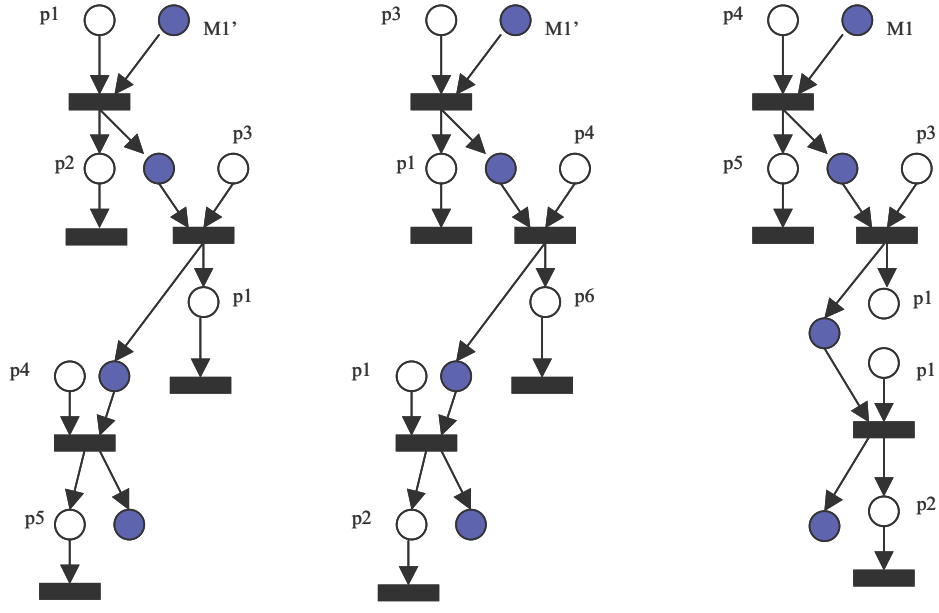


Fig. 7. The five BUCs of M1



(a) A1-A3-B2 (b)A3-B2-A1 (c)B2-A3-A1

Fig. 8. Example of the unfolding of M1

In this net, we can find the 6 processes of M1 are as follows (Fig. 9):
Suf1 = t1 t5t3 (15), Suf2 = t5t1t3 (15), Suf3 = t1t3t5 (12),
Suf4 = t3t1t5 (12), Suf5 = t3t5t1 (15), Suf6 = t5t3t1 (15),
where () is an operation time of Sufi.

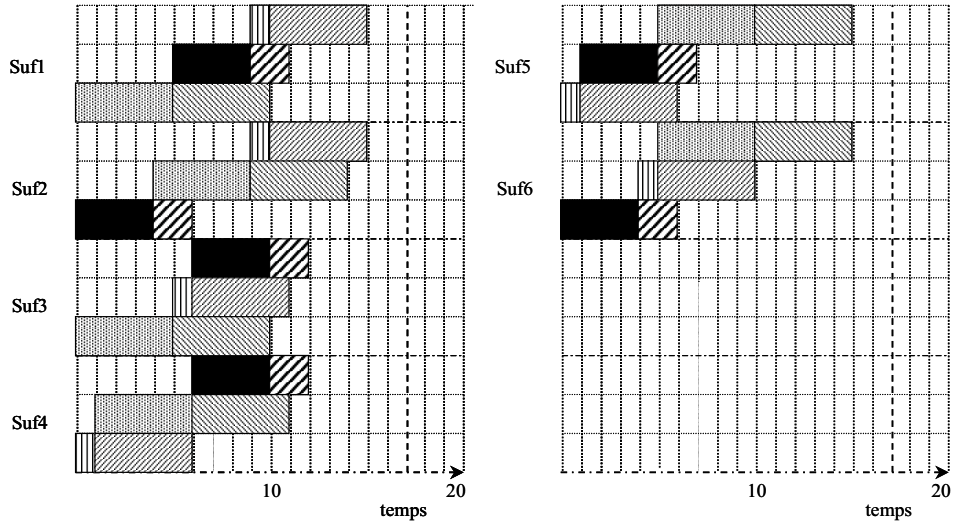


Fig. 9. Results of the permutations of BUC in M1

In M1, we can choose two schedules as transitions sequences: t3-t1-t5 and t1-t3-t5.

2. Modeling of M2 and its unfolding nets

Machine M2 involved two tasks (OP1 and OP2) in three processes (t2, t4 and t6) (Fig. 10,11).

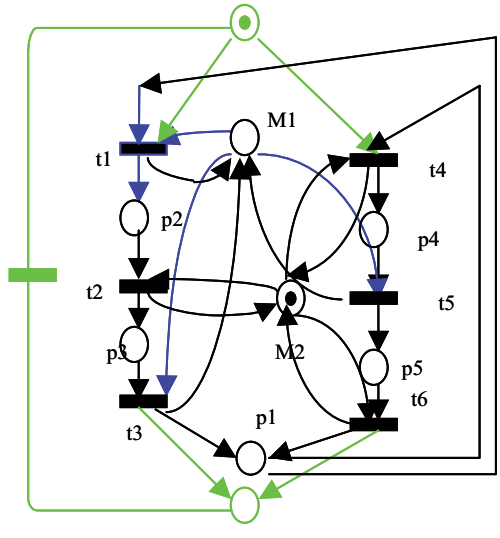


Fig. 10. The BUC of M2

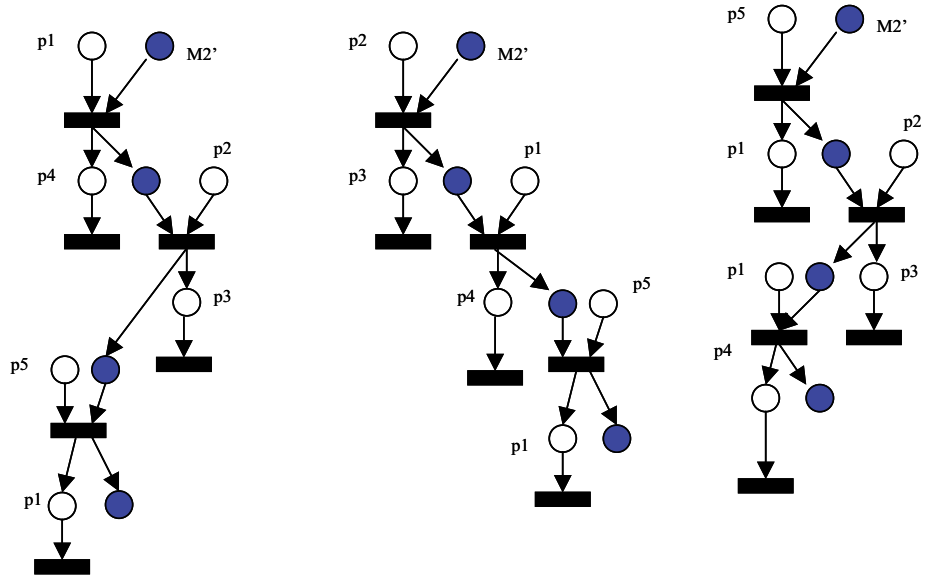


Fig. 11. Example of unfolding of M2

We can show the six processes like as follows (Fig. 12) :

Suf1 = t2t4t6 (13), Suf2 = t2t6t4 (14), Suf3 = t4t6t2 (11),
Suf4 = t4t2t6 (13), Suf5 = t6t4t2 (11), Suf6 = t6t2t4 (13).

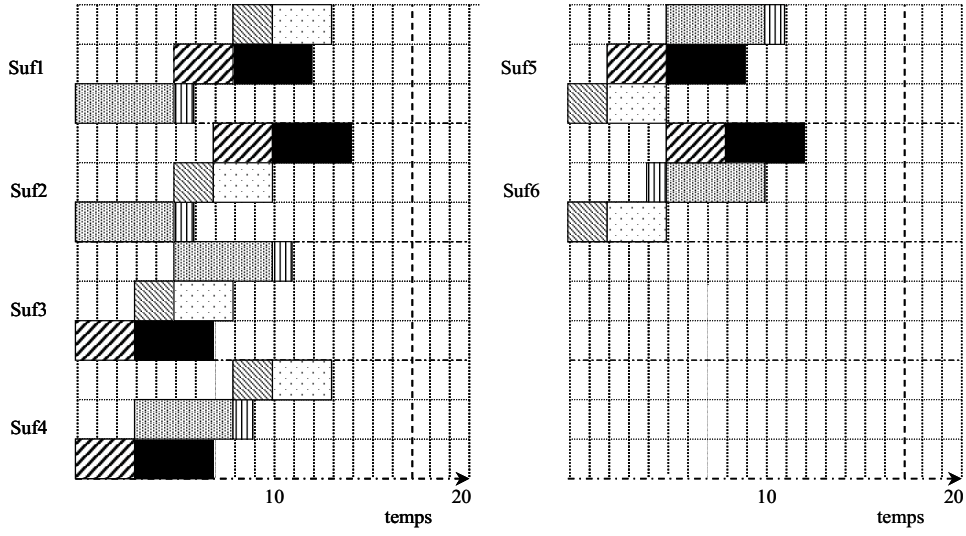


Fig. 12. Results of permutations of BUC in M2

In M2, we can find two solutions like as Suf3 and Suf5. Now, we apply the selected solutions of BUC of M2: {Suf3 and Suf5} to obtain the solution BUC on M1: {Suf3 and Suf4}, then we obtained two solutions. The optimal schedules of two cycles are in Fig. 13 and 14.

Bs1: A3A1B2 (M1) B3B1A2 (M2)

Bs2: A1A3B2 (M1) B3B1A2 (M2)

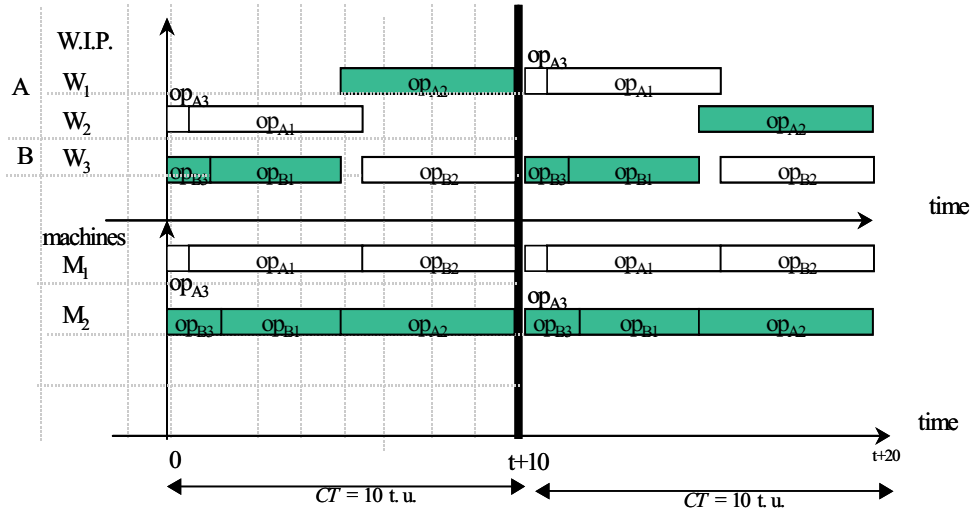


Fig. 13. Optimal schedule of Bs1

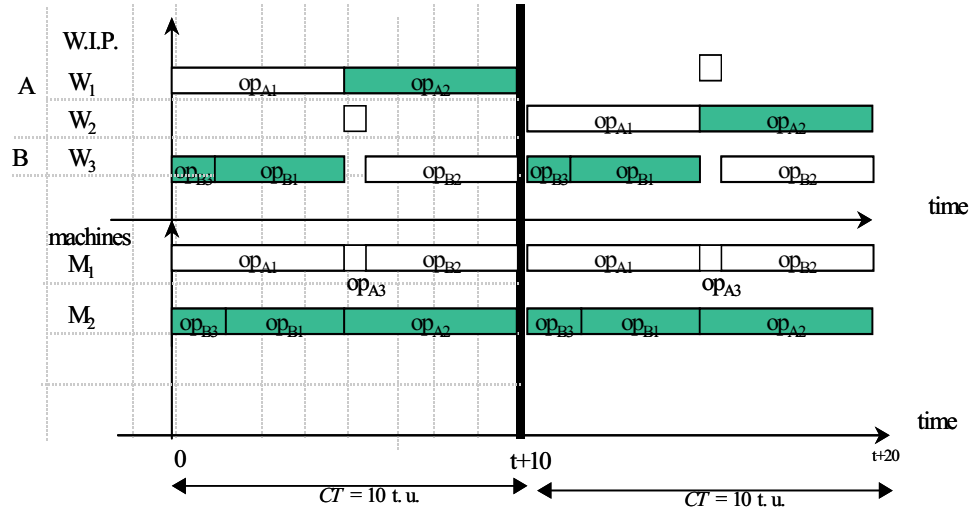
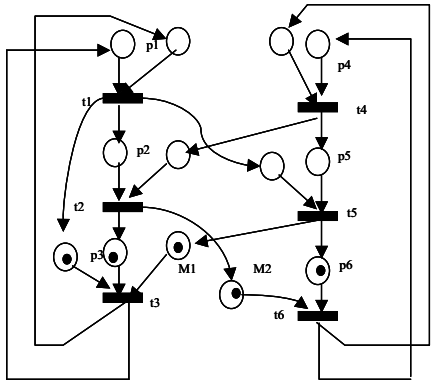
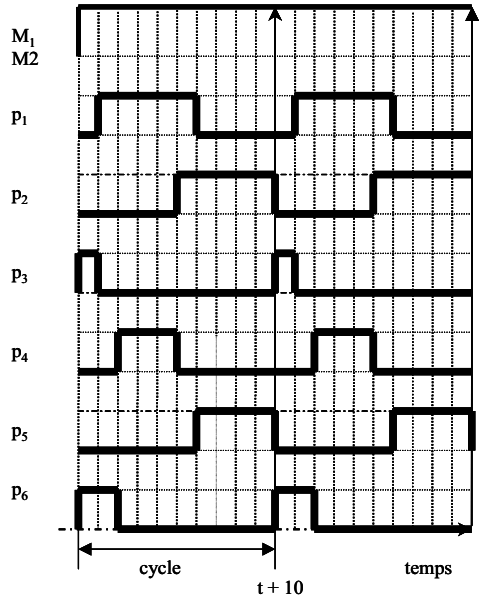


Fig.14. Optimal schedule of Bs2

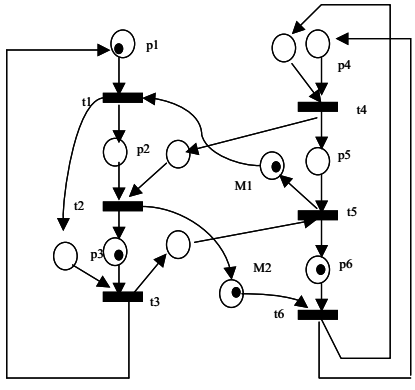


(a) Linear schedule

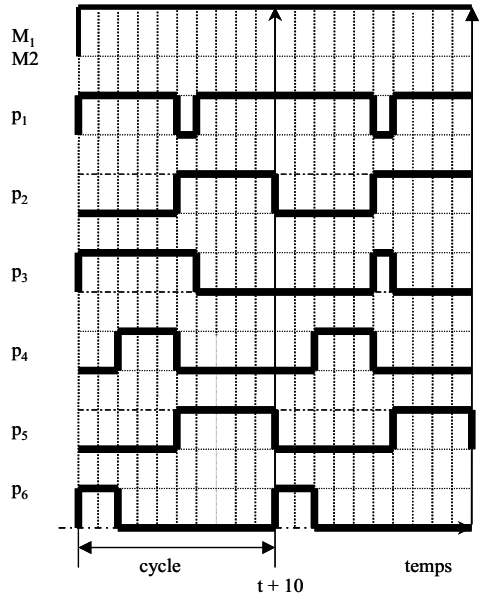


(b) The flow of marking of (a)

Fig. 15. Linear schedule of Bs1



(a) Linear schedule



(b) The flow of marking of (a)

Fig. 16. Linear schedule of Bs2

Finally, we get three pallets rather than two, which is lower bound WIP. Indeed in this model, it is impossible to optimize CT with two pallets, as proved in (Camus,1997). So, we can say that this solution is best possible one(Fig. 15,16).

6. Benchmark

6.1 Notations

In this section, one example taken from the literature is analyzed in order to apply three cyclic scheduling analysis methods such as Hillion (Hillion, 1987), Korbaa (Korbaa, 1997), and the previously presented approach. The definitions and the assumptions for this work have been summarized (Korbaa,1997).
The formulations for our works, we can summarize as follows:

$$\mu(\gamma) = \sum_{\forall t \in \gamma} D(t) \quad ,$$

the sum of all transition timings of γ
$M(\gamma)$ (=$Mo(\gamma)$), the (constant) number of tokens in γ,
$C(\gamma) = \mu(\gamma)/M(\gamma)$, the cycle time of γ,
Where γ is a circuit.
C* =Max($C(\gamma)$) for all circuits of the net,
CT the minimal cycle time associated to the maximal throughput of the system:
CT =Max($C(\gamma)$) for all resource circuits = C*
Let CT be the optimal cycle time based on the machines work, then WIP is (Korbaa,1997):

$$WIP = \sum_{pallets\ type\ i} \left\lceil \frac{\sum Operating\ time\ OS\ to\ be\ carried\ by\ i}{CT} \right\rceil i$$

We introduce an illustrative example in Camus(Camus, 1997), two part types (P1 and P2) have to be produced on three machines U1, M1 and M2. P1 contains three operations: u1(2 t.u.) then M1(3 t.u.) and M2(3 t.u.) P2 contains two operations: M1(1 t.u.) and U1(2 t.u.). The production horizon is fixed and equalized to E={3P1, 2P2}. Hence five parts with the production ratio 3/5 and 2/5 should be produced in each cycle. We suppose that there are two kinds of pallets: each pallet will be dedicated to the part type P1 and the part type P2. Each transport resource can carry only one part type. The operating sequences of each part type are indicated as OS1 and OS2. In this case, the cycle time of OP11, OS12 and OS13 are all 7 and Op21 and OS22 all 3, also the machines working time of U1 is 10, M1 is 11 and M2 is 6. So the cycle time CT is 10. The minimization WIP is:

$$WIP = \left\lceil \frac{\sum \text{Operating Times of } OS_{p1}}{CT} \right\rceil + \left\lceil \frac{\sum \text{Operating Times of } OS_{p2}}{CT} \right\rceil$$

$$= \left\lceil \frac{7+7+7}{11} \right\rceil + \left\lceil \frac{3+3}{11} \right\rceil = 3$$

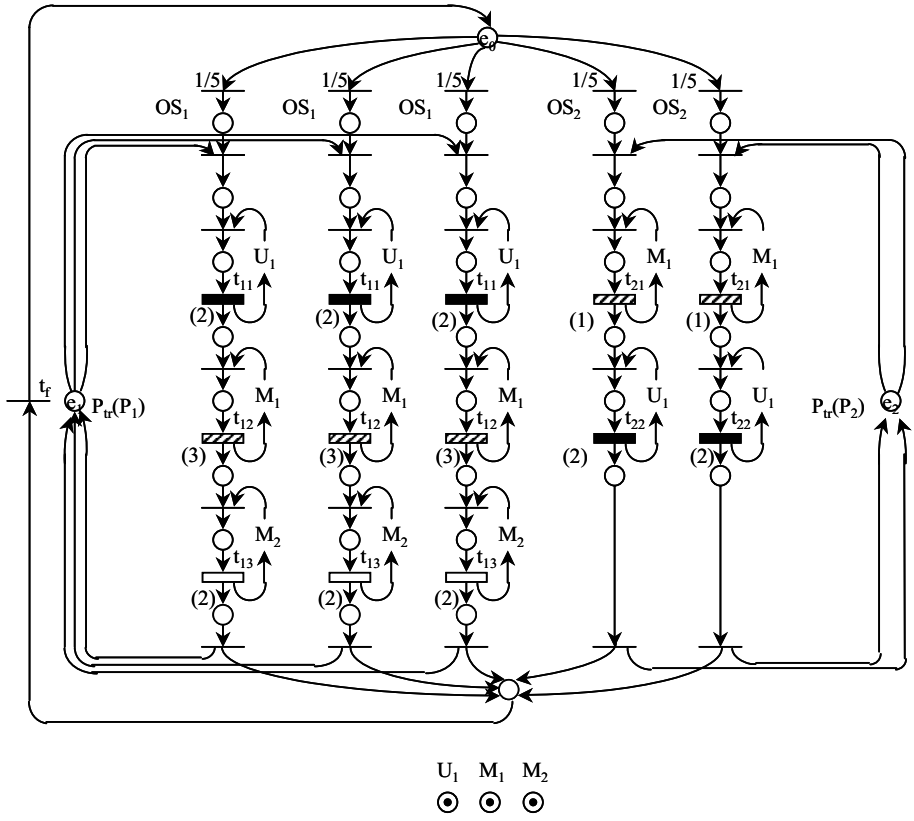


Fig. 17. Illustrative example

6.2 Benchmark

By the example, we can obtain some results like as the following figures (Fig. 18-20).

1. Optimization

The Hillion's schedule (Hillion, 1987) has 6 pallets, the Korbaa's schedule (Korbaa, 1997) 3 ones, and the proposed schedule 4 ones. This solution showed that the good optimization of Korbaa's schedule could be obtained and the result of the proposed schedule could be better than that of the Hillion's.

Also, the solutions of the proposed approach are quite similar to (a) and (c) in Fig. 20 without the different position.

2. Effect

It's very difficult problem to solve a complexity value in the scheduling algorithm for evaluation. In this works, an effect values was to be considered as the total sum of the numbers of permutation and of calculation in the scheduling algorithm to obtain a good solution. An effected value of the proposed method is 744, i.e. including all permutation available in each BUC, and selecting optimal solution for approach to next BUC. An effect value to obtain a good solution is 95 in the Korbaa's method; 9 times for partitions, 34 times

for regrouping, and 52 times for calculation cycle time. In the Hillion's method, an effected value is 260; 20 times for machine's operation schedule and 240 times for the job's operation schedule.

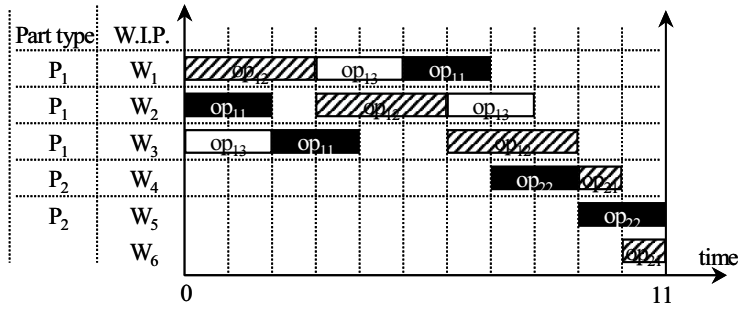


Fig. 18. Hillion's schedule

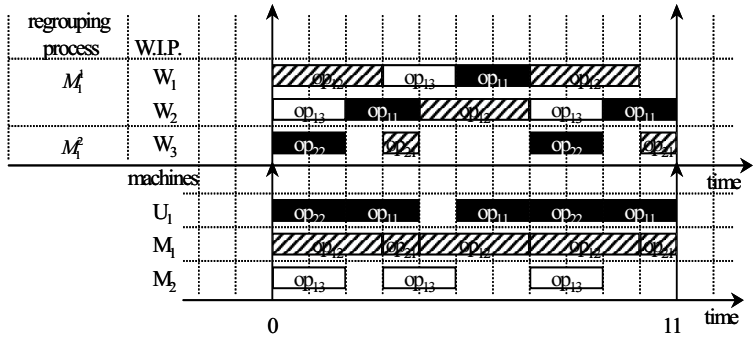
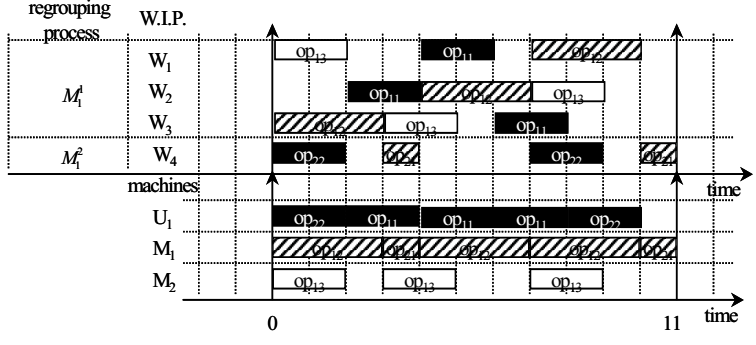
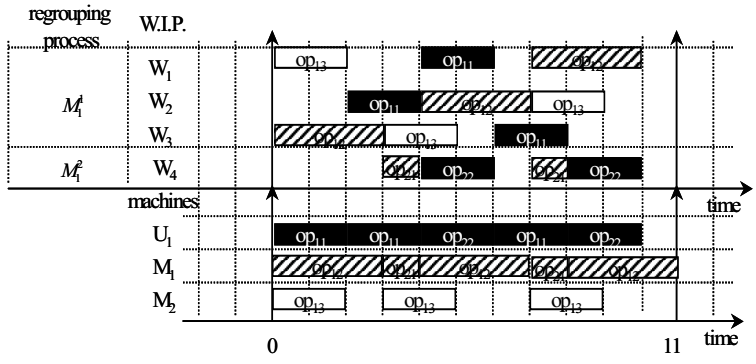


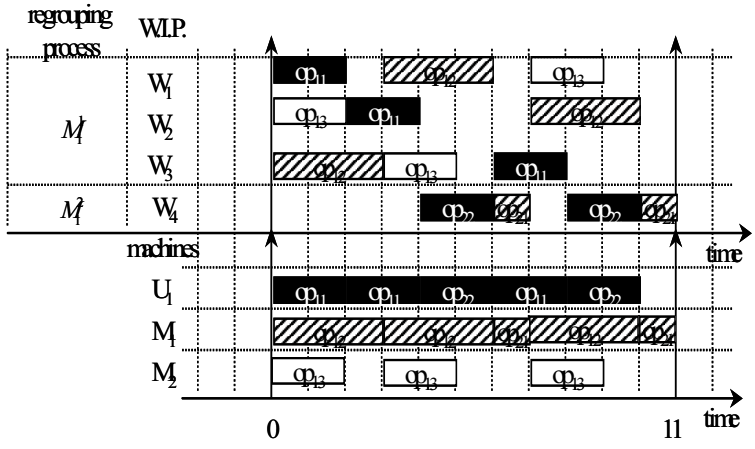
Fig. 19. Korbaa's schedule



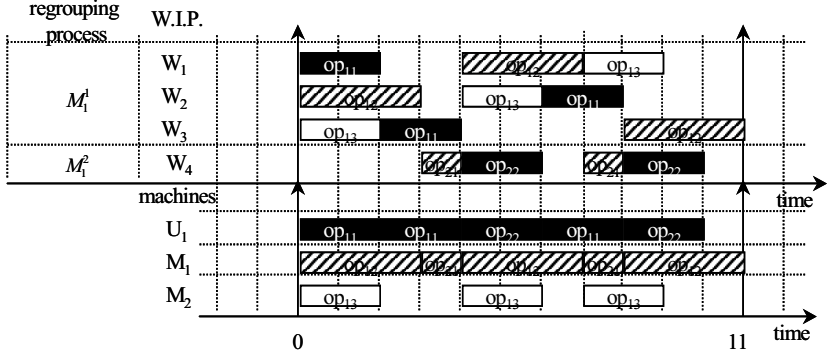
(a)


regrouping process
W.I.P.
W1
W2
W3
W4
machines
U1
M1
M2
time
0
11

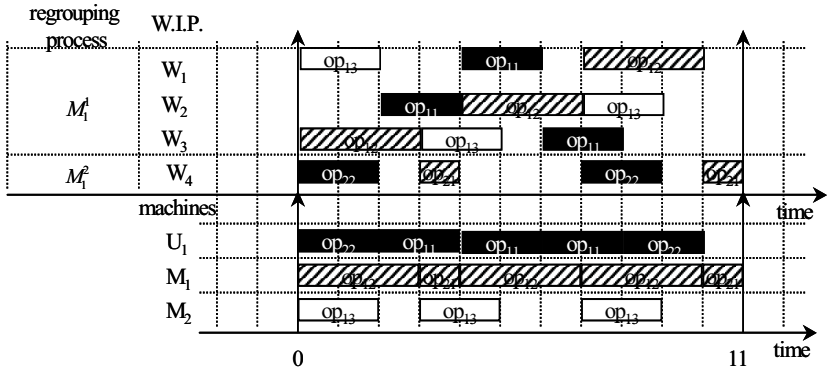
(b)


regrouping process
W.I.P.
W1
W2
W3
W4
machines
U1
M1
M2
time
0
11

(c)


regrouping process
W.I.P.
W1
W2
W3
W4
machines
U1
M1
M2
time
0
11

(d)



(e)

Fig. 20. Proposed schedule

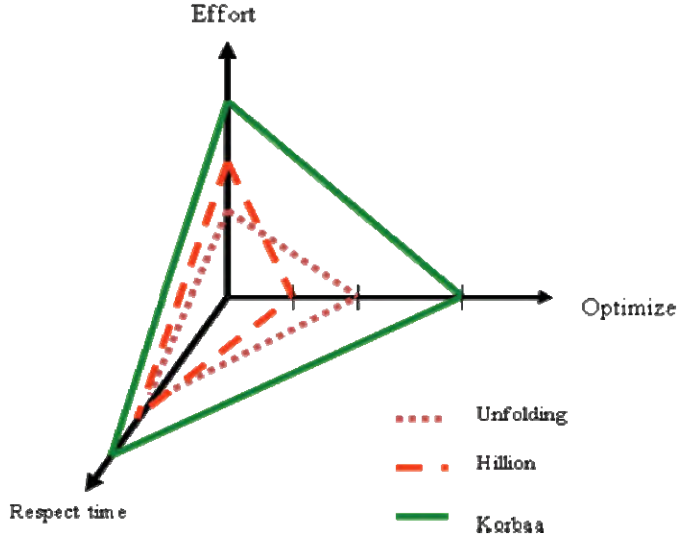


Fig. 21. Total relation graph

3. Time

Based on the three algorithms, we can get time results for obtaining the good solution. Since this example model is simple, they need very small calculation times; 1 sec for the Korbaa's approach and 1.30sec for both of the Hillion's and the proposed approaches. The Korbaa's approach has minimum 1 minute and maximum 23 hours in the 9 machines and 7 operations case in Camus (Camus, 1997), while the proposed approach 3 minutes. Meanwhile the Hillion's and the Korbaa's approaches belong to the number of the operation and the machines, the proposed method to the number of resource shares machines. This means that the Hillion's and the Korbaa's approaches analyzing times are longer than the proposed one in the large model. As the characteristic resultants of these approaches are shown in Fig. 21, the Korbaa approach is found out to be good and the Hillion approach is

to be effectiveness in the time. And on the effort point, the proposed approach is proved to be good.

7. Conclusion and future study

In this paper, we focused on the analysis of a cyclic schedule for the determination of the optimal cycle time and minimization of WIP (Work In Process). Especially, this paper product ratio-driven FMS cyclic scheduling problem with each other products and ratios has been dealt. We proposed a model that has two jobs and two machines. And TPN slice and unfolding are applied to analyze this FMS model. We can divide original system into subsystem using TPN slice and change iterated cycle module into acyclic module without any other behavior properties.

Specially, we simulated our approach with IBM PC windows 2000 using Visual C++, then our approach is faster than Korbaa's approach in the many resource shared. This means that the new approach is more useful to the model that has many resource share machines in any case. If the model has small resource share machines and short operation depths, then it's useful to approach Korbaa's.

We are sure that proposed method is very useful to analyze all Petri net models. This proposed method is available to apply to a complex computer simulation, a parallel computer design and analysis, and a distributed control system, etc.

8. References

Best E., Cherkasova L., Desel J. & Esparza J.(1990). Characterization of Home States in Free Choice Systems, *Hildesheimer Informatik-Berichte* Vol.9/90, Universitat Hildesheim

Carlier J. & Chretienne P.(1988). Timed Petri nets Schedules, In: *Advanced in PN*, G. Rozenberg(Ed.), vol.340 of LNCS, pp.62-84, ISBN 0-387-50580-6,Springer-Verlag,Berlin, Germany

Camus H.(1997). Conduite de Systèmes Flexibles de Production Manufacturière Par Composition de Régimes Permanents Cycliques:Modélisation et Evaluation de Performances à l'Aide des Réseaux de Petri, Thèse doctorat USTL

Esparza J., Lomer S. & Vogler W.(1996). An Improvement of McMillans unfolding Algorithms, IN: *LNCS 1055*, pp.87-106

Hwang CH. & Lee DI.(1997). A Concurrency Characteristic in Petri net Unfolding," *Proceeding of SMC'97*, pp. 4266-4273

Hillion H., Proth J-M. & Xie X-L.(1987). A Heuristic Algorithm for Scheduling and Sequence Job-Shop problem, *Proceeding of 26th CDC 1987*, pp.612-617

Julia S., Valette R. & Tazza M.(1995). Computing a feasible Schedule Under A Set of Cyclic Constraints, *Proceeding of 2nd International Conference on Industrial Automation*, pp.141-146, Nancy 7-9, Juin, 1995

Kondratyev A., Kishinevsky M., Taubin A. & Ten S.(1998). Analysis of Petri nets by Ordering Relations in Reduced Unfolding, *Formal Methods in System Design*, Vol. 12, No.1, pp. 5-38

Korbaa O., Camus H. & Gentina J-C.(1997). FMS Cyclic Scheduling with Overlapping production cycles, *Proceeding of ICATPN'97*, pp.35-52

Lee DY. & DiCesare F.(1995). Petri Net-based heuristic Scheduling for Flexible Manufacturing, In: *Petri Nets in Flexible and Agile Automation*, Zhou MC.(Ed.), pp.149-187, Kluwer Aca. Pub., USA

Lee J.K. & Korbaa O. (2006). Scheduling Analysis in FMS Using the Unfolding Time Petri nets, *Mathematics and Computer in Simulation*, Vol.70, pp. 419-432,

Lee J.K., Korbaa O., & Gentina J-C.(2001). Slice Analysis Method of Petri nets in FMS Using the Transitive Matrix, *Proceeding of INCOM01,* ISBN:0-08-043246-8,Vienna,Austria,Control Problem in Manufacturing, Elsevier Science

Lee J.K. & Korbaa O. (2004). Modeling and analysis of radio-driven FMS using unfolding time Petri Nets , *Computer Ind. Eng.(CIE)*, Vol.46,No.4, pp. 639-653

Liu J., Itoh Y., Miyazawa I. & Seikiguchi T.(1999) A Research on Petri nets Properties using Transitive matrix", *Proceeding of IEEE SMC99*, pp.888-893,

Murata T.(1989) Petri Nets: Properties, Analysis an Applications, *Proceedings of the IEEE*, vol. 77, No. 4, April 1989, pp. 541-580.

McMillan. K.(1995). A technique of state space search based on unfolding, *Formal Methods in System Design* Vol. 6, No.1, pp. 45-65

Ohl H., Camus H., Castelain E. & Gentina JC.(1995). Petri nets Modeling of Ratio-driven FMS and Implication on the WIP for Cyclic Schedules, *Proceeding of SMC'95*, pp.3081-3086

Richard P.(1998). Scheduling timed marked graphs with resources : a serial method, *Proceeding of INCOM'98*

Taubin A., Kondratyev A. & Kishnevsky M.(1997). Application of Petri Nets unfolding to Asynchronous Design, *Proceeding of IEEE-SMC 1997*, pp.4279-4284

Valentin C.(1994). Modeling and Analysis methods for a class of Hybrid Dynamic Systems", *Proceeding of Symposium ADPM'94*,pp.221-226

Zuberek W., Kubiah W.(1993). Throughput Analysis of Manufacturing Cells Using Timed Petri nets, *Proceeding of ICSYMC 1993*, pp.1328-1333

14

Error Recovery in Production Systems: A Petri Net Based Intelligent System Approach

Nicholas G. Odrey
Department of Industrial and Systems Engineering, Lehigh University
USA

1. Introduction

Leading-edge companies require flexible, reliable and robust systems with capabilities to adapt quickly to changes and/or disturbances. In order to be adaptable a flexible manufacturing systems must possess the ability to (i) reconfigure the existing shop floor and (ii) automatically recover from expected and unexpected errors. One of the major problems in flexible manufacturing systems is how to effectively recover from such anticipated and unanticipated faults. Traditional techniques have addressed the error recovery problem from the point of view of defining a set of actions for a pre-specified set of errors. The main disadvantage of this approach is that not only a huge amount of coding is required but also that two undesirable situations still may occur: (i) some errors may not occur in a prespecified set during the lifetime of the system and (ii) there may be errors that cannot be anticipated. Pre-enumerating a large number of error occurrences will not guarantee that the system will not encounter a new error situation. Our intent here is to show the genesis of work into intelligent control of discrete event dynamic systems to overcome (ii) as exemplified by a Petri Net based model for large scale production systems. Petri Nets have been successfully used for modeling and controlling the dynamics of flexible manufacturing systems (Hilton & Proth, 1989; Zhou & DiCesare, 1993). Generally, in a Petri net, the operations required on a part are modeled with combinations of places and transitions. The movement of tokens throughout the net models the execution of the required operations. The content of this chapter is multi-faceted. Topics include Petri Net modeling, state space representation and associated solution techniques, hierarchical decomposition and control, hybrid modeling, multiple agent systems, and, in general, issues pertaining to our work on intelligent control of manufacturing systems.

Our focus here is on the characteristics of physical error occurrences which impose difficult challenges to discrete event control. The majority of our effort has been on workstation/cell control within the hierarchical system originally proposed by the National Institute of Standards and Technology (NIST) e.g. (Albus, 1997). The controller must first handle simultaneously production and recovery activities, and second, treat unexpected errors in real-time to avoid a dramatic decrease in the performance of the system. In the following sections we follow the modeling approach previously presented by (Odrey& Ma, 1995) which had its origins in the work of (Liu,

1993). This previous work included modeling, optimization, and control within the framework of hierarchical systems. In particular, the research was focused on efforts towards the foundations of a multilevel multi-layer hierarchical system for manufacturing control. The Petri Net formalism can handle the complexities of the highly detailed activities of a manufacturing workstation such as parallel machines, buffers of finite capacity, dual resources (multiple resources required simultaneously on one operation), alternative routings, and material handling devices to name a few. Details on the mathematical structure and definitions pertaining to Petri nets can be found in numerous sources e.g., (Zhou & Dicesare, 1993; Murata, 1989). The reader is referred to this literature for detailed underlying mathematical models. A further thrust of our work has been to enhance a multilevel multi-layer model by the incorporation of intelligent agents with the purpose of adding flexibility and agility. Thus, one objective of our effort is to determine whether it is possible to integrate Petri Nets constructs with object-oriented formalisms and have an "all in one" modeling and implementation tool for intelligent agent-based manufacturing systems. Several researchers have attempted to combine these techniques. One of the first approaches was Object Oriented Petri Nets (Lee and Park, 1993).

More recent work pertains to addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system (Odrey & Mejia, 2003). The system consists of production, mediator, and error recovery agents. Production agents contain both planner and control agents to optimize tasks and direct material flow, respectively. Here we address the error recovery agent within a hierarchical system at the workstation level in more detail. It is assumed that raw sensory information has been processed and is available. When an error is detected, the control agent requests the action of a recovery agent through a mediator agent. In return, the recovery agent devises a plan to bring the system out of the error state. Such an error recovery plan consists of a trajectory having the detailed recovery steps that are incorporated into the logic of the control agent. In the context of Petri Nets, a recovery trajectory corresponds to a Petri subnet which models the sequence of steps required to reinstate the system back to a normal state. After being generated, the recovery subnet is incorporated into the workstation activities net (the Petri Net of the multi-agent system environment). In this research, we follow the designation of others (Zhou & DiCesare, 1993) and denote the incorporation of a recovery subnet into the activities net as net augmentation. The terms "original net" or "activities net" refer to the Petri Net representing the workstation activities (within a multi-agent environment) during the normal operation of the system. The net augmentation brings several problems that require careful handling to avoid undesirable situations such as the occurrence of state explosions or deadlocks. Intelligent agents seem to be a promising approach to deal with the unpredictable nature of errors due to their inherent ability to react to unexpected situations. Research on intelligent agents in the context of manufacturing have been mostly concentrated on the "production activities" e.g. scheduling, planning, processing and material handling (Gou, et al., 1998; Sousa & Ramos, 1999; Sun, et al., 1999) However the activities related to exception handling such as diagnostics and error recovery have received little attention. Our research aims to provide some evidence as to how the performance of a manufacturing system can be improved by using intelligent agents modeled with Petri Nets.

1.1 Statement of the problem

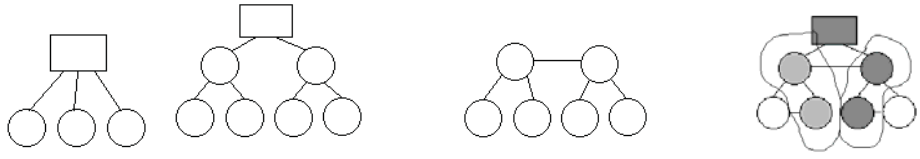
The focus in this chapter is on physical error occurrences and is directed towards supporting effective procedures for error recovery in an attempt to arrive at a reconfigurable, adaptive, and "intelligent "manufacturing system. As such, a hybridization of Petri Nets and intelligent agents seem to be a promising approach to deal with the unpredictable nature of errors due to their inherent ability to react to unexpected situations. Within this context, we investigate system learning with a hybrid Petri net-neural net structure. The following sections of this chapter first discuss the background on architectures for reconfigurable and adaptable manufacturing control. Subsequent discussions will be based on the genesis of work at Lehigh University on Petri nets from initial modeling and solution approaches to more recent work on embedding intelligent agents with Petri Nets. A hybrid nets consisting of a Petri Net with a Neural Net approach for the purpose of intelligent control is also discussed.

2. Architectures

Even though our focus in this chapter is on Petri Net modeling and error recovery , we would be remiss to not mention the underlying architecture of the systems being investigated, While some performance tests (Brennan, 2000; Van Brussel, et al.,1998) suggest that intelligent agent architectures for manufacturing systems outperform other architectures, the lack of standards on design methodologies, communication protocols and task distribution among the agents makes difficult their introduction to real-life applications. Opposed to intelligent agent-based architectures, hierarchical architectures have been conceived with the standardization issues in mind. A hierarchical architecture groups the elements of the manufacturing system into hierarchical levels, e.g. enterprise, factory, shop, cell, manufacturing workstation and equipment levels, with the purpose of coping with complexity. The major drawback of hierarchical architectures is that their structure is overly rigid and consequently difficult to adapt to unanticipated disturbances (Van Brussel, et al., 1998). To increase the functionality of the system, components at the same level may be linked. The purpose was to loosen the strict master-slave relationship of the proper hierarchical form. This resulted in the so termed, modified hierarchical form. Higher flexibility was reported with this architecture; however some problems arose in the communication links between entities of the same level mostly caused by the lack of development of the technology available at that time (Dilts et al., 1991).

To overcome the difficulties of the hierarchical architectures a heterarchical (distributed) form was proposed (Duffie et al., 1988). In this architecture a single entity did not exist at the top level as in the hierarchical scheme. In this architecture there existed a number of parts or components which "negotiate" the utilization of scarce resources. As such, a feedback signal did not have to go one level up in the hierarchy to find a response and a corrective action. A system failure in the context of this architecture meant "lack of communication" between two entities. As one communication link failed other resources were capable of establishing the linkage. There was not a single information source as the information was distributed throughout the system. Ideally the system would have been very flexible and adaptable as new elements (software or hardware) could have been "attached" to the existing ones without major disruptions. The heterarchical control architectures coped very well with disturbances and reacted quickly to changes but the lack of hierarchy led to unpredictability in the system. Consequently global optimization was almost impossible because there was

neither global information nor a higher-level entity that controlled the overall performance of the system. Responses to perturbations that could be assimilated to "quick fixes" or expediting could have caused further disturbances. Further developments led to the concept of holonic manufacturing (Van Brussel et al., 1999; Valckernaers et al., 1994). The Holonic paradigm considers three primary (basic) types of agents: Order agents, product agents and resource agents, each with different goals and functionality. The basic agents are assisted by other specialized agents namely staff agents which take the role of higher-level controllers in a hierarchy (Van Brussel et al., 1999). These staff agents are at fact in a higher level of the hierarchy but their role is only to provide expert advice to the basic agents instead of enforcing rules. To tackle with complexity and to avoid a large number of low-level agents trying to interact, agents are grouped and classified into categories. An agent is dual entity that is both a part and an autonomous entity. Related agents form aggregated agents as in a hierarchical structure but that structure differs from the traditional approach which aims for a fixed structure. The holonic hierarchy is loosely connected. This means that the configuration of the system can be changed to adapt to new conditions (Bongaerts, 1999). The ease of adaptation implies a high degree of compatibility and ex-changeability between the software and hardware elements of the system. The following figure depicts the structure of different control architectures. Notice that in the Holonic model, the modules can be reconnected and form new hierarchies. The basic elemental structure of the discussed architectures is sketched in Figure 1.



a) Hierarchical b) Heterarchcal c) Holonic

Fig. 1. Basic Control Architectures

The architecture adopted in our research consists of a multi-agent system inspired by the holonic architecture developed in Europe (Van Brussel et al., 1999) and the elementary loop function (ELF) modified from the work at NIST for intelligent systems (Meystel & Albus 2002; Albus & Barbera, 2005). It has been noted that the ELF architecture is common to most intelligent systems (Meystel & Messina, 2000). In essence we are attempting to capture and implement the flexibility, adaptability, and reconfigurability required for an environment (production systems modeled via Petri nets) subject to various disturbances. A later section provides more detail as to the status of this work.

3. Workstation modeling

3.1 Workstation modeling with alternative routing

Earlier research on Petri Net modeling and analysis at Lehigh University was focused on a hierarchical structure for automated planning and control of a cellular-based shop. (Liu et al., 1997; Odrey and Ma, 1995) The adopted architecture was a hierarchical structure that followed a model developed by Saleh (1988) that was based on the hierarchical model of the National Institute of Standards and Technology (NIST) (Jones and McLean, 1986). Saleh's model incorporated both multi-levels and multi-layers. Multi-levels were designed to

partition the complex structure of the shop into smaller decision and control units such as shop, cell, workstation and equipment levels. In this research we developed three different layers of control, namely the optimization, regulation and adaptation layers. The purpose was to develop a near-optimal steady state schedule along with the corresponding regulatory actions in the event of disturbances.

Following Saleh's work, Liu (1993) constructed a Timed Colored Petri Net (TCPN) model for a manufacturing cell. A three attribute coloring scheme was used and is described later. One example of a cell contained two workstations; the first workstation was a material handling device and the other described a loading/unloading station. This is shown in Figure 2 on the next page. For brevity, only a partial description of all places is given. The objectives here were (i) the construction of a PN model with rerouting capabilities, and (ii) the development a state-space representation to predict and optimize the dynamics of this system. To model a flexible manufacturing cell a machine oriented approach was undertaken and was based on modular constructs. This approach provided a construct such that a sudden addition or reduction of system resources (e.g., machines) required a minimal restructuring of the workflow within the production system. It should be noted that it can still take a great amount of effort for modeling of a PN based system. The TCPN cell model in this earlier research was determined by the system capacity of the cell and the production workload. The system capacity included the number of workstation types, the number of parallel resources in a workstation, and the material handling system (MHS). The production workload included job types, the processing times, and the routing of jobs.

From a Petri net viewpoint the system capacity dictated the configuration of the cell model whereas the production workload determined the number of job tokens and operational circuits in the workstation subnets. Figure 2 depicts a TCPN for the system but note that the recovery from machine breakdowns is not included in this figure. Two job types were modeled in the cell. The two workstation subnets and the load/unload (L/UL) subnet are connected in parallel through the MHS subnet. The parallel subconnections subnets fulfilled a requirement of a random direction material flow. The interface between cell entities are the two sets of places {P4, P7, and P15} and {P27, P25, and P26} which represent the input queues and output queues to the L/UL station and workstations W1 and W2, respectively. In this model the number of tokens in each closed-loop subnet represented the total availability of a particular resource in a cell entity. For example, two tokens in place P9 represent two identical machine resources in workstation W1, whereas a single token in places P6 and P12 represent a single space for the input and output buffer, respectively, of workstation W1. In a TCPN cell model, token colors are useful for both visual identification and mathematical representation. Consider place p1 in Figure 2. Two job types identified by their different token colors (one black dot and a white circle pattern). In the case of parallel resources, i.e., two parallel machine tokens in P9, distinctive colors would be used for individual resource identification.

In this modeling approach, a three-attribute coloring scheme (part number, workstation number, resource number), was used to differentiate token colors. Part number (pt#) represents the job number; Workstation numbers (wks#) indicates the workstation where a part is currently being processed or is to be processed; a resource number refers to either a buffer number (b#) or a machine number (m#) in a particular workstation, an equipment number (e#) in a load/unload station, or a device number (d#) for material handling systems. These resource attributes provide a tracking record for the resource assignment decisions. Hence, a token color, (i, j, m), indicates that the token is the i^{th} job which uses the

m^{th} resource in the j^{th} workstation. The coloring scheme is embedded in the matrix representation of the TCPN cell model used in the system dynamic equations.

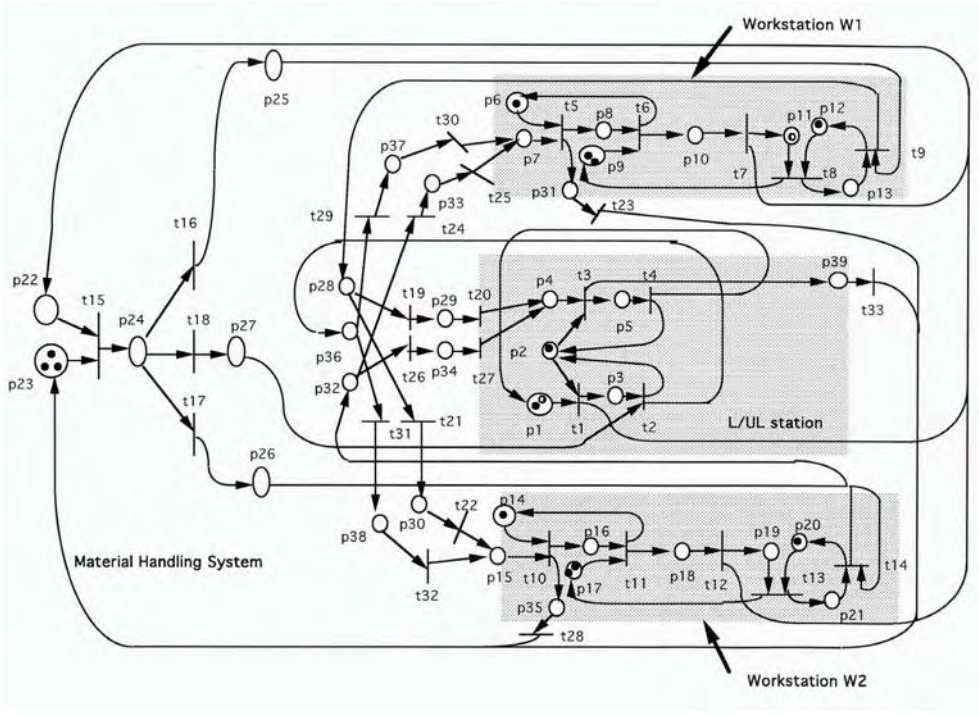


Fig. 2. Time Colored Petri Net for Two Workstations a load/Unload station, and a Material Handling System (Liu, et al. 1997; Liu, 1992)

In this research, the modular construct was a convenient restructuring method proved adaptable to changes in the production environment. The possible system configuration changes were categorized into two types: changes in a physical entity or changes affecting jobs. In the event of adding or deleting a physical entity (e.g., a workstation), the workstation subnet was connected or disconnected to/from the MHS subnet. In this earlier work, if machine breakdowns occurred the corresponding machine resource token was simply stopped from circulating in the subnet until recovery. For entity disruption, the overall model structure remained relatively the same. Any changes affecting jobs consisted of a cancellation of jobs or changes in the job routing information. Job routing changes involved the deletion of operation circuits from previous stations and the addition of operation circuits to the new stations. Furthermore, in this research, for each physical entity considered in a cell there existed a 1-to-1 representation in the TCPN model. As such, each operation performed had a corresponding processing time associated with the operation circuit in the processing workstation and each system resource corresponding token representation. Parallel resources were represented by multiple resource tokens of the same color in the cell model. The total number of token types represented the total number of that resource types available in the system. This TCPN development methodology provided a safe, bounded, and live model.

Naturally, an important consideration was the representation of a disruption (error) occurring and a possible rerouting strategy. This was approached by noting that machine breakdowns can be satisfied by regarding a machine breakdown as an external input (the firing of a transition in a Petri net model). This additional structure provided an immediate transfer of tokens from a place (which represents processing) without waiting for the elapse of processing time. Figure 3 depicts a TCPN workstation which can be used to represent an alternative routing logic for machine breakdowns.

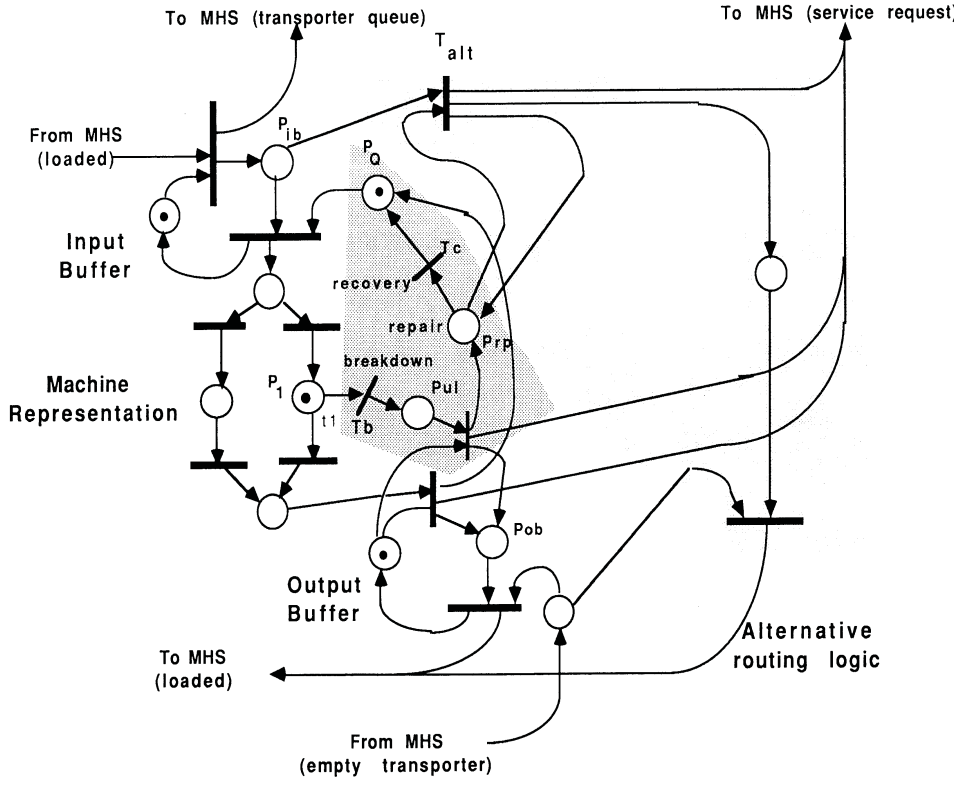


Fig. 3. A Workstation Petri Net Representation with an Alternative Routing Logic (Liu, 1993)

Firing transition Tb represents the fact that a machine breakdown has occurred. The token in place P1 is released to Pul. In such an instance the remaining processing time (t1) is set to zero. This unload place may have a queue and waits for an output buffer to unload the part from the breakdown machine. In this representation 3 tokens are generated once an output buffer is generated. A machine token passes to a repair process Prp whereas a token in place Pr signals a service request for the material handling system (MHS). Simultaneously a job token is outputted to place Pob (place signifying an output buffer). Other transitions noted in Figure 3 consist of Talt (initiate re-rote mechanism to alternative machines) and Tc (to indicate recovery of machine from the breakdown). The firing of transition Tc causes the machine token to be returned to the common queue (place P_Q) and stops the firing of the alternate machine transition Talt. At the time this scheme was developed to overcome

drawbacks associated with 1) an inhibitor arc approach (Teng & Black, 1990) and, 2) a timed Petri net representation by (Barad & Sipper, 1998). An inhibitor arc approach cannot provide a systematic mathematical representation in the event of changes in transition firing rules. The work here was a modification of the latter TPN approach.

3.2 Workstation analysis

The state space representation used to analyze the workstation Petri nets was a modification of the traditional state equation (Murata, 1989) with the incorporation of equations for the remaining processing times of every timed place. The conventional state space representation can be written as:

$$\boldsymbol{M}(k+1) = \boldsymbol{M}(k) + \boldsymbol{L}\,\boldsymbol{u}(k)(1)$$

where $\boldsymbol{M}(k)$ is the marking of the Petri Nets in time k, $\boldsymbol{L}$ is the incidence matrix and $\boldsymbol{u}(k)$ is the vector of transition firings. The reader is referred to Murata (1989) and Al-Jaar and Desrochers (1995) for details on this equation.

The state space representation developed by Liu (1993) considers operational, precondition, post-condition and resource places. Only operational places (those where actions are carried out) have associated processing times. The other places, as

their name suggest, represent conditions (e.g. idle, ready) (Liu et al., 1997). The modified structure contains two different "marking" vectors: the first marking vector ($\boldsymbol{M}_p(k)$) is the conventional marking vector (Murata, 1989) that accounts for the number of tokens in each place; the second one ($\boldsymbol{M}_r(k)$) is the remaining processing time vector i.e. a vector containing the remaining time for the next transition firing for each place.

The state space equation is stated as follows (the dimensions of these matrices are omitted for simplicity):

$$\boldsymbol{X}(k+1) = \boldsymbol{A}(k)\,\boldsymbol{X}(k) + \boldsymbol{B}(k)\boldsymbol{u}(k) \tag{2}$$

$$\boldsymbol{X}(k) = \begin{vmatrix} \boldsymbol{M}_p(k) \\ \\ \boldsymbol{M}_r(k) \end{vmatrix} \tag{3}$$

u(k) is a control vector that determines which transitions fire at time k. Define uj(k) as the jth position of u at time k. uj(k) = { 1 if transition j fires, 0 if it does not } Mp(k) is the marking vector at after k transition firings; Mr(k) is the remaining processing time vector after k transition firings; A(k) is the system matrix and it is partitioned as follows:

$$\boldsymbol{A}(k) = \begin{vmatrix} [\mathrm{I}] & [0] \\ \\ -\tau(k)[\mathrm{P}] & [\mathrm{I}] \end{vmatrix} \tag{4}$$

[0] Zero matrix;

[I] Identity matrix

τ(k) Time for the next transition firing.

[P] Diagonal matrix that serves to distinguish operational places from resource, precondition and post-condition places.

Pii = {1 if place pi is an operational place; 0 otherwise}

Pij = 0 when i≠j

B(k) is the distribution matrix that transforms the control action u(k) into token evolution i.e. addition and removal of tokens when firing a transition represented in vector u(k).

$$\boldsymbol{B}(k)=\begin{vmatrix} & [L] \\ [W] & [L]^{+} \end{vmatrix} \tag{5}$$

[W] = Processing time matrix for operational places.
[L]= Incidence matrix [L] =[L]+ - [L]-
[L]+ = Incidence output matrix that accounts for the addition of tokens in output places.
[L]- = Incidence input matrix that accounts for the removal of tokens from input places.
The dimension of these matrices is determined by the number of places, transitions and colors in the system. For a detailed discussion and explanation see (Liu, 1993; Liu et al., 1997). This representation was the basis for an optimal control formulation for scheduling optimization. A near-optimal solution was found by using forward dynamic programming on the sequence of states (markings) generated by the state equations.

3.3 Petri Net Decomposition

In the process of establishing a hierarchical Petri net-based workstation model, issues can be categorized into different classes where each class occurs at different levels of the hierarchy.

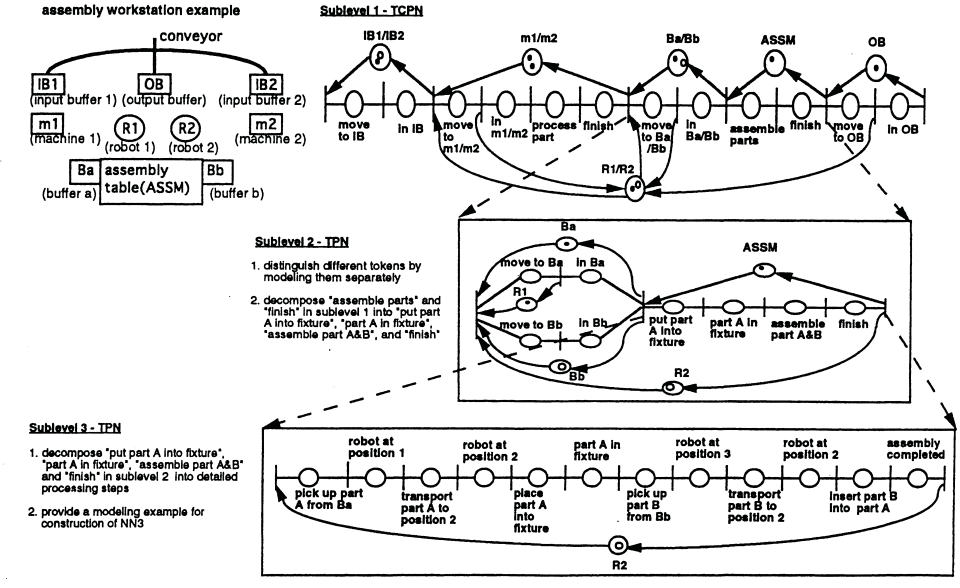


Fig. 4. An example of decomposition of a multi-layer Petri net model for an assembly station (Ma & Odrey, 1996)

At the Petri net modeling level two decision classes were identified, namely, generation of conflict-free sequences and the determination of process steps sequences. In order to facilitate the decision-making and performance evaluation processes, a hierarchical system

of state equations for the Petri nets based model was studied. The general form of the hierarchical state equations have previously been state in equations 2 through 5. An example of the net decomposition for an assembly workstation is indicated in Figure 4. For the top level TCPN model (termed sublevel 1), the state dimension depends on three values: (1) the sum of all colors on tokens associated with places which represent the process of manufacturing individual parts, (2) the sum of all colors on tokens associated with places which represent the process of handling assembled final products, and (3) the sum of all colors on tokens associated with the resource places. When decomposing the TCPN model to a sublevel 2 TPN model the system can be viewed as a two-level hierarchical Petri net with one discolored TPN at the upper level and several subnets, which are also modeled by TPNs, at the lower level. Between upper and lower levels, interface places are added that serve as connectors between two levels. For a state space representation, the discolored TPN at the upper level and each detailed subnet at the lower level can be individually represented using TPN state equations. Thus, the system state equations for the sublevel 2 TPN workstation model are obtained by combining all the TPNs and augmented to incorporate the interface places, i.e. all the vectors/matrices in the subnets are become the subvectors/submatrices in the sublevel 2 TPN workstation state equation. For example, the distribution matrix for the sublevel 2 TPN model would have the form of the matrix given below. L^i is a distribution submatrix of TPNi .The bottom row denotes the distribution submatrices of the interface places and the input/output transitions associated with TPNi. Details of this work can be found in (Odrey & Ma, 2001). This multi-level, multi-layer Petri net framework establishes layers to provide the linkage between high-level abstract information for discrete systems and

$$L = \begin{bmatrix} L^1 & [0] & [0] & \cdots & [0] \\ [0] & L^2 & [0] & \cdots & [0] \\ [0] & [0] & L^3 & \cdots & [0] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ [0] & [0] & [0] & \cdots & L^J \\ L_c^1 & L_c^2 & L_c^3 & \cdots & L_c^J \end{bmatrix} \tag{6}$$

low-level numeric data for continuous systems. Different nets are used to represent different levels of complexity. Three functional distinct subnets which are the basic building blocks for the Petri net workstation model were proposed to represent higher level abstract commands such as “move,” “process,” and “assemble”. These subnets allow basic routing information to be incorporated in the model through a bottom-up approach in a systematic manner. The process task can then be decomposed into a Petri net representation of process steps which follow a feature-based process plan. Alternative sequences and resources are incorporated in the process task model to provide flexible operation instructions. Dynamic state space equations correspond to each sub-level in the hierarchical Petri net graphical representation. These state equations are used in current research to evaluate various control strategies and performance workstation operations in a unifying way.

4. Intelligent system approaches using Petri nets

4.1 Intelligent agent approaches

Current efforts are directed towards the aspects of error recovery associated with intelligent agent-based manufacturing systems and has been motivated by the work done at Lehigh

University. As noted above, previous work included modeling and optimization and control of hierarchical systems. Our focus is to enhance this multilevel multi-layer model with the in-corporation of intelligent agents with the purpose of adding flexibility and agility. This on-going effort investigates (i) architecture reconfigurations with enhanced capabilities of flexibility and adaptability, (ii) the adoption of adequate model-ing techniques and their mathematical representation (in particular, modifications to the previous Timed Colored Petri Net models developed), (iii) modeling the aforementioned intelligent agents with Petri Nets, and (iv) model testing.

Our motivation has its origins in the research mentioned in the previous sections in addition to models incorporating intelligent agents for manufacturing operations which appeared in the eighties and nineties as an alternative to the shortcoming of hierarchical and heterarchical architectures. Some of these additional approaches include Bionic Manufacturing (Okino, 1993), Fractal methods (Warnecke, 1993), the MetaMorph Architecture (Wang et al., 1998; Maturana et al., 1998). These approaches preserve a hierarchy that controls the autonomy of individual agents, but unlike the hierarchical architectures, the relation-ship between low and high level controllers (agents) does not follow the master-slave scheme. The low level agents have a high degree of autonomy as in the heterarchical approach but still have "loose" links with higher-level agents. An intelligent agent based approach attempts to preserve the advantages of both hierarchical and heterarchical approaches but at the same time avoids their drawbacks. The architectures mentioned present differences primarily in the definitions of the intelligent agents, the degree of reactivity versus long-term planning, the degree of adaptation and reconfiguration, and the communication methods between agents. For example, in the Holonic, Bionic, MetaMorph and Fractal approaches the intelligent agents are loosely connected and their structure can evolve over time; the RCS resembles a hierarchical architecture whose structure is primarily fixed. In the Holonic, MetaMorph and RCS approaches the system has a set of fixed predefined goals. In the Fractal approach the agents negotiate their goals (Tharumarajah et al., 1996). Bionic architectures (Okino, 1993) do not set long-term goals but seek essentially adaptation to the environment. In the Holonic manufacturing approach parts, computers and resources are considered as intelligent agents. The other approaches regard schedulers, planners, controllers and resources as agents, but exclude parts.

It should be noted that the concept of Intelligent Agents was built around the Object-Oriented Programming (OOP) paradigm (Tharumarajah et al., 1996). The underlying principle of OOP is the encapsulation of attributes and methods into code units called classes. The code embedded in a class defines its internal actions and the relationships with other classes (Wyns and Langer, 1998). In the intelligent agent approach, each agent becomes an object with clearly defined functionality and attributes. Thus these concepts of OOP such as instantiation, inheritance, and polymorphism can be applied directly to the theory of intelligent agents (Venkatesh and Zhou, 1998). To date OOP platforms are the preferred choice for control software development (Gou et al., 1998). Some of its advantages over conventional programming include reusability, portability and expandability. OOP seems to be the natural approach to implement the control software for intelligent agent-based architectures (Gou et al., 1998). Venkatesh and Zhou (1998) have pointed out need for integration of control and simulation and modeling software to expedite the system development. In other words, the control software should not be exclusively dedicated to issue commands to the components of the manufacturing systems but to optimize the system performance. It should also be noted that all agents are objects but not all objects are

agents. Agents are autonomous entities that have choices and control on their behavior; objects may be totally obedient (Jennings, 2000).

4.2 Multi-agent systems with embedded Petri nets

Our more recent work presents an architecture for control of flexible manufacturing systems which is a synthesis of hierarchical and intelligent agent-based systems (Odrey & Mejia, 2003). The approach undertaken provides responsive and adaptive capabilities for error recovery in the control of large scale discrete event production systems. A major advantage of this is the ability to reconfigure the system. The communication links between agents can be re-directed in order to form temporary clusters of agents without modifying the internal structure of the agent. At the same time, having the hierarchical structure greatly facilitates the organization of new groups of agents. In our approach, agents possess the freedom to move within their hierarchical level but cannot move out to another level. The approach, based on Petri Net constructs is expected to improve the performance of agent-based systems because (i) it decentralizes the control activity for complex and unusual failure scenarios (ii) provides basic autonomy to resource agents (iii) follows a proved design hierarchical design methodology and, (iv) defines clearly the responsibilities of control and resource agents. A thrust of this effort was to determine whether it is possible to integrate Petri Nets constructs with object-oriented formalisms and have an "all in one" modeling and implementation tool for intelligent agent-based manufacturing systems.

At the time of this investigation the major focus was on the diagnostics and error recovery activities in the context of intelligent agent-based architectures for semi-automated or autonomous manufacturing systems. Our approach addressed the issue of combining the discipline of hierarchical systems with the agility of multi-agent systems. We adopted in-part the holonic paradigm (Van Brussel et al., 1999) for description of the three primary (basic) types of agents: Order agents, product agents and resource agents, each with different goals and functionality. The basic agents are assisted by other specialized agents namely staff agents which take the role of higher-level controllers in a hierarchy. In particular, the focus was on the construction of a re-configurable system having production agents, error recovery agents, and a classifier/coordinator/ mediator agent structure connecting production and recovery agent hierarchies. In addition, the relationship to the previous work at Lehigh University pertaining to a multi-level, multi-layer hierarchy control was established. This latter hierarchy, based on Petri net constructs, serves, in one sense, as a retrieval based resource for process planning and generation of re -cover plans to the production and recovery agents within the proposed multi-agent system. An objective of this effort was to provide a test-bed for comparison of purely hierarchical systems, non-hierarchical but highly re-configurable multi-agent systems, and a hybrid combination which was the focus of the investigation presented here. Our primary efforts are on a hierarchical intelligent agent-based system linked to a structure of agents dedicated exclusively to diagnosis and error recovery tasks. Our work has focused primarily on error recovery strategies at the workstation level in an intelligent-agent based system and is still on-going.

Unlike the traditional structure (Albus, 1997) in which the control function is exerted top down, our approach provides the agents basic control capabilities that allow them to react to common and local disturbances. In addition, specialized control and recovery agents assist these production agents on complex diagnostics and recovery tasks. This approach is expected to combine the discipline of hierarchical systems, but with the inherent ability to react as would be congruent with intelligent agent-based systems. Here we adapt the

intelligent agent principles to hierarchical control models. The most significant difference lies in the definitions of a workstation controller and a workstation agent. In a broader scope, a workstation agent comprises a workstation controller, a number of resources (conveyors, machines, tools, fixtures, etc.) and their respective controllers [Van Brussel, 1998]. The workstation agent acts as a single decision unit when negotiating with higher-level agents. At the same time a workstation agent is considered as a system when controlling and coordinating its components (equipment agents and error recovery agents).

4.2.1 System structure

Our approach is based on prior work on hierarchical architectures as outlined in previous sections. As such our model shares a number of features with the prior work, namely, hierarchical decomposition of activities, sensor strategies, methods for diagnosis and error recovery, and modeling techniques. The reader is referred to (Odrey & Mejia, 2003) for a more detailed explanation of this section. A sketch of the integrated control architecture is shown in Figure 5. The architecture is partitioned into three segments. A mediator agents structure is positioned between and separates a production agents architecture and a recovery agents architecture. Each of these structures follows a hierarchy and communication can be at and among different levels within the hierarchy. To-date, we distinguish between cell level and workstation level production agents which communicate through mediator agents. In the schema adopted if an error occurs at the shop floor and the workstation agent cannot produce a satisfactory recovery plan by itself, such an agent requests the actions of the workstation mediator agent. The workstation agent provides all the available information pertaining to the error which should include sensor readings, location, priority, etc. The mediator agent classifies the error and matches the error with a recovery agent at the same hierarchical level. The recovery agent attempts to produce a recovery plan and if it succeeds the plan is communicated back to the mediator. At the same time, if the error exceeds a pre-determined time threshold, the workstation agent sends a message to the cell agent (higher level) informing of the abnormality. The cell agent takes this new input and determines whether or not rescheduling pending jobs is necessary. In order to keep the system running, the workstation agent adopts a temporary measure e.g., dispatching rules. At this point, this is the maximum the workstation agent can do since it lacks of the information and methods to perform global optimization. When a new schedule, generated by the cell agent, is available, the workstation agent attempts to adapt the new plan to the current conditions. In this way, each agent contributes independently to the overall optimization of the system.

The workstation agent requires additional techniques to optimize the realization of the process plan of all the current jobs that have been allocated to it. In our approach, the workstation agent itself constructs a Petri Net model of the sequence of coordinated activities for all current jobs using a multi-level multi-layer Petri Net approach [14]. In this approach the sequence of activities at the workstation level and the required resources are modeled using several "layers" which represent degrees of modeling abstraction (from generic activities to highly specific tasks). As noted in the previous section the highest layer is modeled with a Timed Colored Petri Net (TCPN). The TCPN layer is then "unfolded" in several layers with different degrees of detail. Lower levels are represented by Timed Petri Nets and Ordinary Petri Nets. For each of these nets in order to track the system status state equations can be developed. These equations serve to determine the flow of tokens and the remaining process times for each operation place provided by a sequence of transition firings.

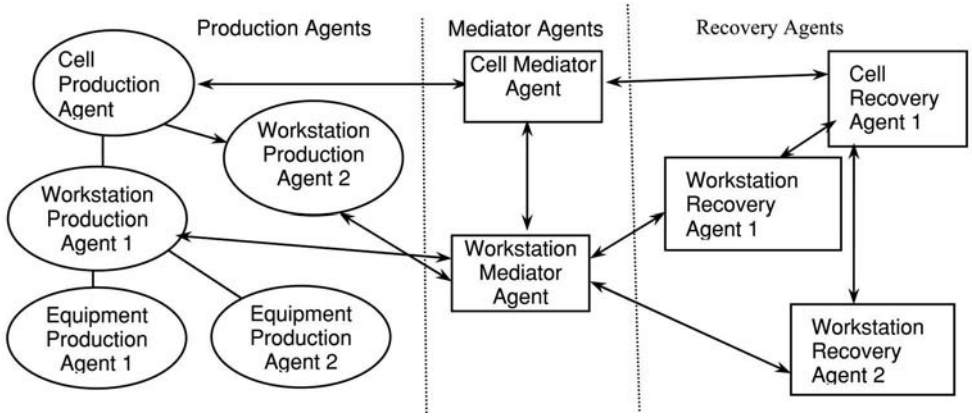


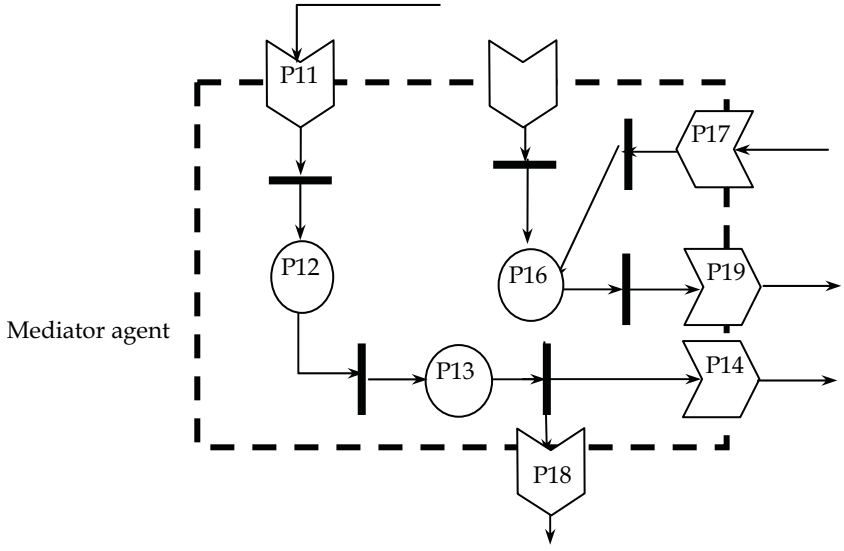
Fig. 5. Error recovery agents within an intelligent agent hierarchical architecture (Odrey& Mejia, 2003)

The BRIC (Block-like Representation of Interactive Components) was chosen as our initial modeling tool in that adoption serves very well to develop control software in that it provides the most important features of OOP (Object Oriented Programming). Additionally, BRIC provides a graphical representation of the behavior of a multi-agent system. In the BRIC approach each agent is modeled by a Petri subnet that comprises an internal net representing the agent's methods and a set of "communication" places. Agents are linked together by through external transitions and arcs. Tokens flowing between communication places serve as message between agents. The complex data structure is embedded in the colored token coloring scheme. For example, a token in an input message interpreted as a work order could include several different labels such as sender id, job priority, job constraints , etc. Conventional token rules of Colored Petri Nets (CPN) apply to the communication places. A token can go from a conventional place to a communication place and vice-versa. For further details the reader is referred to (Odrey & Mejia, 2003). It should be noted that the agent interaction/communication structure is an on-going investigation. Other techniques are currently being investigated.

4.2.2 Mediator agent

Mediator agents are the link that connects the production structure with the recovery structure. Their function is to facilitate the communication between production and recovery agents. The primary functions of mediator agents are: 1) filtering/processing sensory information from production agents, 2) classifying errors and performing preliminary diagnostics based on feedback information, 3) matching errors that occur on the shop floor with error recovery agents, and 4) communicate recovery plans to production agents. A BRIC model of the structure of a mediator agent is shown in Figure 6. Places are as defined. In this schema, a mediator agent first receives a request (P11) and classifies the error (P12) according to a set of corrective preliminary actions. We adopt here the approach of our previous work (Ma, 2000) in which error classification was performed using a Petri Net embedded in a neural network linked to an expert system. Next, a matching module embedded in the recovery agent attempts to match the error with recovery agents capable of generating a recovery plan for the error that occurred (P13). The issue of matching errors

with recovery agents is a subject of further research. If a suitable recovery agent is found, the mediator sends it a request for recovery (P14). A token in P15 represents that a recovery plan (or a failure to generating a plan) has been received. The mediator agent evaluates the received plan (P16) and communicates it to the corresponding production agent (P19). Provisions are made should the mediator agents require aid from other mediators (places P17 and P18).



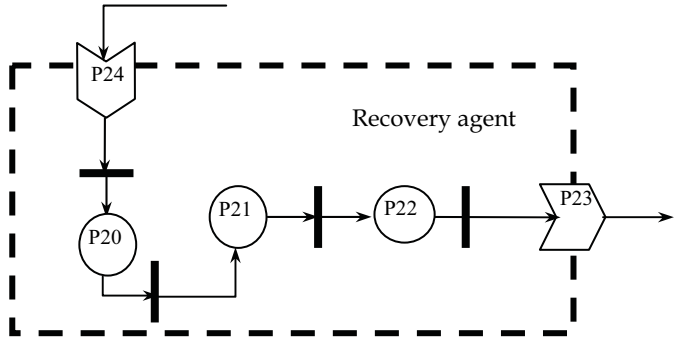
P11: Receiving recovery request message
P12: classifying errors
P13: Matching classified error with recovery agents
P14: Sending request messages
P15: Receiving responses from recover agents
P16: Evaluating responses
P17: Receiving responses
P18: Sending messages to mediators
P19: Communicating recovery plans

Fig. 6. A BRIC model of Workstation Mediator Agent (adapted from Odrey & Mejia, 2003)

4.2.3 Error recovery agents

The recovery agents at the workstation level are responsible of three major tasks: (i) screening recovery requests sent by mediators, (ii) performing in-depth diagnosis, and (iii) generate recovery plans for expected and unexpected errors. The BRIC model of a workstation recovery agent and place definitions are shown in Figure 7. Once an error is classified a token is placed in P20 and further diagnosis is performed when a marking

reaches P21. When a root cause is known and classified, a plan can be generated (P22) and sent to the appropriate agent via P23 and P24.
Our current efforts here focus on developing an automated reasoning technique for generating recovery plans. The recovery plan generation primarily depends on whether or not the error has been anticipated. Anticipated and unanticipated errors require two different strategies: In the case of anticipated errors, a recovery plan is generated by matching the error with a recovery task in a lookup table (Odrey & Ma, 1995). Unexpected errors require more complicated (deep) reasoning that implies finding and matching error patterns with gross recovery plans or searching alternative paths to return or advance the system to an error-free state. Previous work at Lehigh University (Ma, 2000) was concentrated on generation of gross recovery plans using Neural Networks. The last stage of modeling our proposed architecture consists of linking the agents to form a Petri Net model of the control structure and can be found in (Odrey & Mejia, 2003).



P20: Evaluating classified error
P21:Diagnosing errors (in-depth Diagnosis)
P22: Generating recovery plan
P23: Sending recovery plan to mediator agents
P24: Receiving recovery requests messages

Fig. 7. Workstation Recovery Agent Structure (Odrey & Mejia, 2003)

5. Error recovery approaches

Error recovery is the set of actions that must be performed in order to return the system to its normal state (Odrey and Ma, 1995; Seabra-Lopes and Camarinha-Matos, 1996). The key concept is that there should exist at least one sequence of actions to bring the system to its normal operation. The purpose of error recovery is to find the best actions that minimally disrupt the system while down-time is minimized. Our work presented here follows 2 approaches: 1) the first section used an augmented Petri Net approach and 2) a subsequent section was an attempt to provide a hybrid net by joining Neural Nets with Petri Nets. This was done for a workstation level controller with in a hierarchical system following the work done at NIST. Both of these approaches are discussed in subsequent sections.

5.1 Definitions for diagnostics and error recovery

An *error* occurs when the observed behavior conflicts with the desired behavior of the system (Odrey and Ma, 1995; Seabra-Lopes and Camarinha-Matos, 1996). Similarly, (Chang et al. 1991) defined that an error occurs when a resource reaches an undesired state. (Kokkinaki & Valavani, 1996) define errors as manifestations of faults. A *fault* is the cause of an error (Chang et al., 1991). As long as the error is not detected or does not produce a failure, it remains latent. A *failure* occurs when a re-source does not deliver a service. For example a worn gear in an automated fixture prevented a part to be accurately positioned on a machine tool. Because of this, the part could not be correctly machined and resulted in a bad assembly. The worn gear is the fault that generated the errors and failures. The error is a positional error (the undesired state) and the failure is the wrong assembly (a service that could not be delivered). *Diagnostics* is the activity in which the source fault(s) is (are) determined and isolated (Odrey and Ma, 1995). When a failure is detected, the operation that failed is not necessarily the source of the failure. A source fault is propagated through the system generating errors and failures. Diagnostics involves backtracking the failed operations to the source fault. The failure propagation tree is the tool that serves the backtracking actions by linking operations until the one that failed is found (Chang et al., 1991). In our research incorporating a multi-agent approach faults are considered as inconsistencies in the behavior or status of an agent or inconsistent interactions between agents and between agents and the environment. The environment is everything outside the boundaries of the intelligent agents. For example, a broken gear that produces paralysis in the machine spindle is an abnormal behavior of a resource agent; an out-of-tolerance part is an abnormal state of a part agent; failure to grasp a part is an inconsistent interaction between the robot agent and the part agent and blocking a robot agent by an external entity is an undesired interaction between the robot agent and the environment. When faults occur the workstation controller agents and the low level agents that depend on the workstation controller, namely machine and part, investigate the reasons of the failure. The low level agents investigate their own internal failures and the workstation controller investigates its own internal faults and the interactions between the part and machine agents and between those two and the environment. For now, the work has been focused on Petri net approaches.

5.2 Augmented Petri net approach for error recovery

The approach taken here was based on integrating Petri subnet models within a general Petri net model for a manufacturing system environment, and, in particular, a workstation controller. In essence, the error recovery plan consists of a trajectory (Petri subnet) having the detailed recovery steps that are then incorporated into the workstation control logic. The logic was based on a Timed Petri Net (TPN) model of the total production system. The Petri subset models consist of a sequence of steps required to reinstate the system back to a normal state. Once generated, the recovery subnet is incorporated into the Petri net model of the original expected (error free) model. The workstation controller is the entity responsible for the coordination, execution and regulation of the activities at the physical workstation. The workstation controller receives a higher level command, generally form a higher level controller that issues a set of operations to be performed by the workstation with desired

start and finish times. The workstation controller decomposes such a command into a lower level set of coordinated activities. In addition to executing activities, the workstation controller should also provide a reactive and adaptive response to errors and other disturbances (Odrey and Ma, 1995). In this work we followed the modeling approach discussed in previous sections. The following discussion is a summary of (Odrey and Mejia, 2005).

5.2.1 Relationship to previous work

The characteristics of physical error occurrence impose difficult challenges to the workstation controller. The controller must first handle simultaneously production and recovery activities, and second, errors that appear unexpectedly must be treated in real-time to avoid a sudden decrease of performance. Examples of automated reasoning systems for error recovery procedures, such as neural nets include the work of (Seabra-Lopes et al., 1996; Kokkinati and Valavanis, 1996) and our work discussed in section 5.3. As previously discussed (section 4), work addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system consisted of production, mediator, and error recovery agents. Production agents contain both planner (scheduler) and control agents. In this section we address the error recovery agent within the hierarchical system at the workstation level in more detail. It is assumed that raw sensory information has been processed and is available. When an error is detected, the control agent diagnoses the error and requests the action of a recovery agent via mediator agents discussed in section 4.2.2. In return, the recovery agent devises a plan to bring the system out of the error state. Such an error recovery plan consists of a trajectory having the detailed recovery steps that are incorporated into the control agent logic. A forward trajectory is the most desirable, but at the same time it is the most difficult to implement with automated reasoning systems (Fielding, et al., 1987). In the context of Petri Nets, a recovery trajectory corresponds to a Petri subnet which models the sequence of recovery steps required to reinstate the system back to a normal state. A schematic of error recovery trajectories is given in Figure 8 as follows:

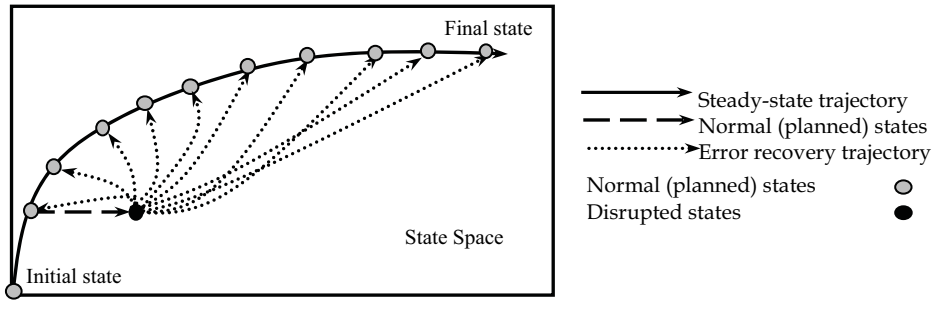


Fig. 8. Error recovery trajectories from a disrupted state (Odrey and Mejia, 2005)

Figure 8 illustrates a view of the issue of "match-up" state in a manufacturing system and shows a desired "trajectory" constructed out of normal states, a disrupted state and the

possible transient trajectories (dotted lines) to return to the original trajectory. The disrupted state is reached involuntarily. After being generated, the recovery subnet is incorporated into the workstation activities net (the Petri Net of the multi-agent system environment). In this research, we followed the designation of others (Zhou and DiCesare, 1993), and denoted the incorporation of a recovery subnet into the activities net as net augmentation. Zhou and DiCesare developed a formal description of these three possible trajectories in terms of Petri net constructs, namely input conditioning, backward error recovery, and forward error recovery. This prior work on error recovery strategies was intended to model the specifics of low level control typified by the equipment level of a hierarchical control system. The terms "original net" or "activities net" refer to the Petri Net representing the workstation activities (within a multi-agent environment) during the normal operation of the system. In the work presented here, the three recovery trajectories are applied to the workstation level within a hierarchical model. The enormous number of errors and the corresponding ways to recover that can occur at the physical workstation implies unlimited possibilities for constructing recovery subnets. The important issue is that any error and the corresponding recovery steps can be modeled with any of the three strategies mentioned above. Without loss of generality, this research limited the types of errors handled by the control agent to errors resulting from physical interactions between parts and resources (e.g. machines and material handling devices). The reason for this assumption was to facilitate the simulation of generic recovery subnets. Backward recovery suggests that a faulty state can become a normal state if an early stage in the original trajectory can be reached. The forward recovery trajectory consists of reaching a later state which is reachable from where the error occurred.

5.2.2 State equations and recovery subnets

The state space mathematical description was briefly described in section 3.2.In general that work consisted of a cell level timed, colored Petri nets (TCPN) state space representation for systems with parallel machining capability. This TCPN state representation extended Murata's generalized Petri net (GPN) state equations by modifying the token marking state equations to accommodate different type of tokens. In addition, a new set of state equations was developed to describe time-dependent evolution of a TCPN model. As a result, the system states of a cell level TCPN model were defined by two vectors:

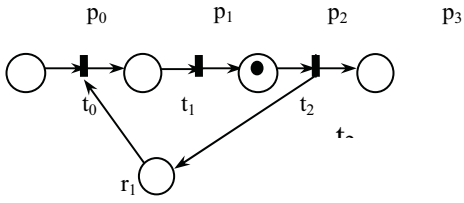
- System marking vector (M^p): This vector indicates the current token positions. A token type may consist of a job token, a machine token, or a combined job-machine token.
- Remaining processing times vector (M^r) : This vector denotes how long until a specific job, machine, or job-machine token in an operation place can be released (i.e. an operation is completed)

The TCPN workstation state equations provide a mathematical evaluation of the workstation performance at a higher level. After evaluation, a decomposed Timed Petri net (TPN) can then be constructed according to the evaluation results along with more detailed workstation operations. This was illustrated in section 3.3. As previously noted, subnets are viewed as alternative paths to the discolored TPN. The alternative path approach taken here is more flexible than a substitution approach in the sense that changes in subnets can be made without changing the configuration of the discolored TPN. The TPN workstation state

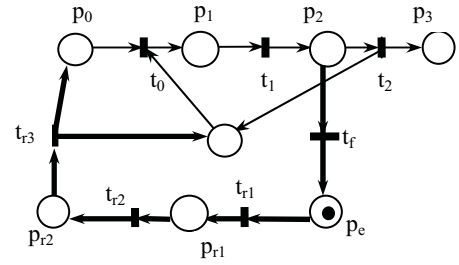
equations provide a mathematical evaluation of the workstation performance at a lower level where primitive activities are coordinated to achieve desired task assignments.
In the event of disruptions, the original activity plan devised off-line by the workstation controller may require adjustments. The question that arises is how to re-construct the activity plan. A first alternative would be to build a completely new plan to execute the pending jobs. The other extreme would be waiting until the disturbance is fixed and continuing with the original plan. This would be partially constructing a new plan to a point where the original plan can be resumed. In terms of the Petri Nets this corresponds to find a marking (state) in the original plan reachable from the disrupted state and the question to be answered is the selection of a marking that should be reached. From there, a number of possibilities exist to return to the original plan. Details on performance optimization are given in a companion paper (Mejia & Odrey. 2004).
In terms of the Petri Nets, an error occurs when a transition fires outside a predetermined time frame. When a transition fires earlier or later (if the transition fires at all) than expected, an alarm is triggered and an error state is produced. After the error is acknowledged and diagnosed, a recovery plan is generated. This is accomplished by linking an error recovery subnet to the activity net. This linking produces an augmentation of the original net. At this stage the controller must devise a plan to reach the final marking M_f based on the status of the augmented net. Reaching the final marking M_f is accomplished by constructing a plan to reach some pre-defined intermediate marking M_{int} from previously determined List markings and then firing the pre-determined sequence of transitions from such an intermediate marking to the final marking. If a path to the intermediate marking can be found, then the original execution policy (sequence of transition firings) can be employed from the desired intermediate marking M_{int} to reach the final marking M_f. The issue of selecting the appropriate intermediate marking can be found in companion article (Mejia and Odrey, 2004). Our focus at this juncture is to demonstrate the construction of recovery subnets.

5.2.3 Construction of recovery subnets for error recovery

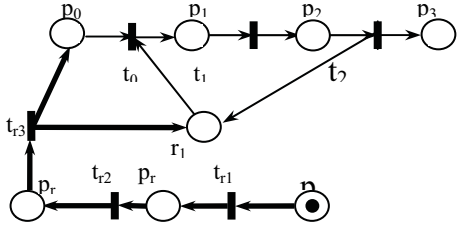
Perhaps the most complete descriptions of error recovery trajectories were developed by (Zhou and DiCesare, 1993). They proposed three possible trajectories. These consisted of input conditioning, forward error recovery, and backward error recovery. Input conditioning notes that an abnormal state can transform into a normal state after other actions are finished or some conditions are met. Forward error recovery attempts to reach a state reachable from the state where the error occurred. Backward error recovery suggests that a faulty state can become a normal state if an earlier stage in the trajectory can be reached. Obviously, not all trajectories are applicable in all cases due to logical or operational constraints. An example demonstrating backward error recovery is presented here but note that a similar approach can be applied to the other types of trajectories. Figure 9 illustrates the events during an error occurrence and the corresponding recovery in terms of Petri Net constructs. Figure (9a) represents the Petri Net during the normal operation. Places are defined in Figure10. The error is represented by the addition of a new transition t_f and a place p_e representing the error state in (9b). Firing t_f removes the residing token in p_2, resets the remaining process time corresponding to the place p_2, and puts a token in the new place p_e. The error recovery subnet and procedure are discussed in more detail in the following section.



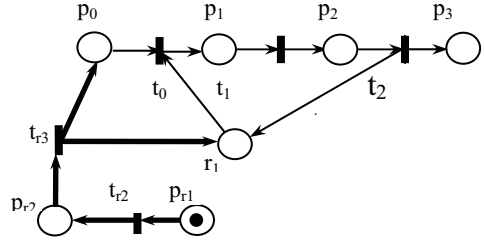
(a) Petri Net of during normal operation. A part is being processed by resource r_1



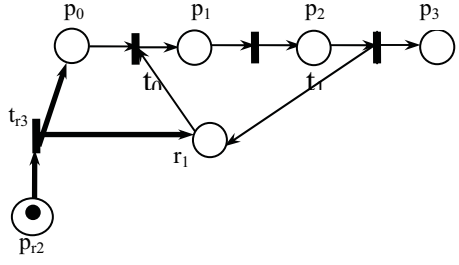
(b) Incorporation of an error/error recovery net. The error/error recovery net is shown with thicker lines.



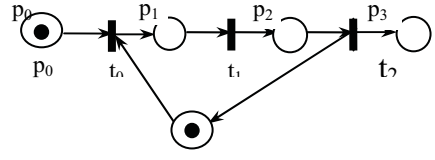
(c) Firing and deletion of t_f and the arc I(t_f, p_e)



(d) Firing and deletion of tr1, the place p_e and the corresponding arcs.



(e) Firing and deletion of t_{r2}, the place p_{r1}, and the corresponding



(f) Firing and deletion of t_{r3}, the place p_{r2}, and the corresponding arcs.

Remarks:

p_e represents an error state.

t_f is the transition that represents the initiation of the failure

p_0 to p_3 represent arbitrary operational places;

p_{r1} and p_{r2} represent recovery steps

t_{r1} to t_{r3} represent the start and end of the recovery step

t_0 to t_2 are changes of events in the original net

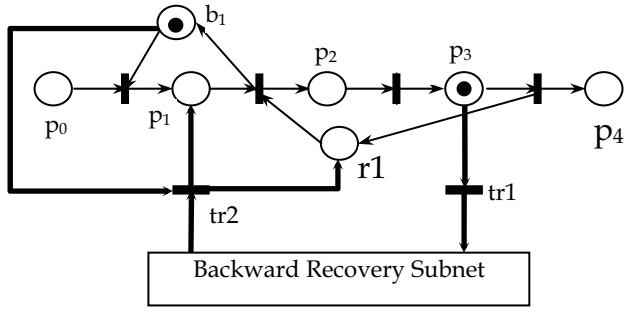
Fig. 9. Construction and Deletion of Recovery Paths (from Odrey and Mejia, 2005).

5.2.4 Incorporating a recovery subnet into the original Petri net

The incorporation of the recovery subnet into the original net by the recovery agent is the first step. In the preceding example (see Figure 9), such a subnet trajectory consists of two places (p_{r1} and p_{r2}) and three transitions (t_{r1} to t_{r3}). Place p_{r1} represents the recovery action "find part" and place p_{r2} the recovery action "pick up part". Transitions t_{r1} to t_{r3} represent the change of states of these two recovery actions. With the recovery trajectory incorporated into the original net, the workstation control agent is required to execute the recovery actions. In (9.b), returning to the normal state requires the firing of transitions t_{r1},t_{r2} and t_{r3}. After firing t_{r3} the scheduled transition firings in the original net resume. The augmented net now contains an Operational Elementary Circuit (OEC) = {p_2, t_f, p_e, t_{r1}, p_{r1}, t_{r2}, p_{r2}, t_{r3}, p_0, t_0, p_1, t_1, p_2} that has only operational (timed) places.

One difficulty that arises is the potential that the operational elementary circuits constructed can result in infinite reachability graphs which make a search strategy difficult. Our approach to overcome this problem consisted of a sequential methodology which eliminates arcs and transitions from the combined original net and error/error recovery subnet. Every time that a transition on the recovery subnet fires, such a transition, its input places (except those places belonging to the original net) and the connecting arcs are eliminated from the augmented net. As noted in Figure 9, the elementary circuit which would be created during the generation of the recovery subnet will only be partially constructed. For example, in (9b), as soon as the transition t_f fires, the transition t_f and the arc I (p_2, t_f) are removed from the net. Subfigures (9c) to (9f) illustrate the sequence of firings and elimination of transitions, places and arcs from the net. The original net is restored when the last transition (t_{r3}) of the error recovery subnet has been fired. After firing t_{r3}, the part token returns to the original net and the resource token to the resource place. The workstation control agent records the elements (places, transitions and arcs) that belong to the original net and recovery subnets, respectively. A record is kept by the workstation controller such that for every time that a transition of the augmented net fires the controller searches for such a transition on the agenda. If the transition is found, it means that the transition belongs to a recovery subnet and all the transition input places and all its input and output arcs are deleted from the recovery agenda and from the augmented net (with the exception of arcs and places belonging only to the recovery subnet and not to the original net).

The next step relates to resuming the normal activities after an error is recovered. In terms of Petri Nets this implies finding a non-error state where the activities net and the recovery subnet are linked. The desired non- error state may not the same as the state prior to the occurrence of the error. For example, the state (marking) in subfigure (9f) is not the same as the state shown in subfigure (9a). The example described illustrates a possible trajectory (backward trajectory) which "started" (according to the arc directions) at p_2. Defining the non-error state is the task of the recovery agent and depends primarily on the characteristics of the error and its recovery. In the event of an input-conditioning strategy, the corresponding net originates and terminates at the same place (Zhou and DiCesare, 1993). Our investigations assume that any part token that goes through either a backward or a forward recovery trajectory is placed in a storage buffers after an error is fixed. Figure 10 illustrates an example for backward error recovery.



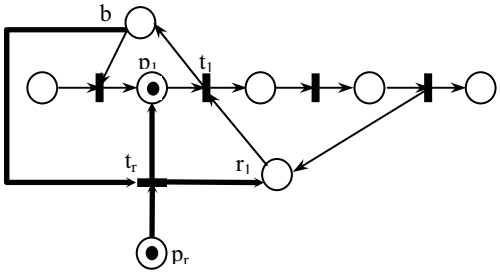
Description of places and transitions
p_0: part available
p_1: part in buffer 1
p_2: part being moved to resource 1
p_3: part being processed by resource 1
p_4: part processed
r1: resource 1 available
b1: buffer 1 available
t_{r1} and t_{r2}: Recovery transitions

Fig. 10. Example of backward recovery trajectory with buffer

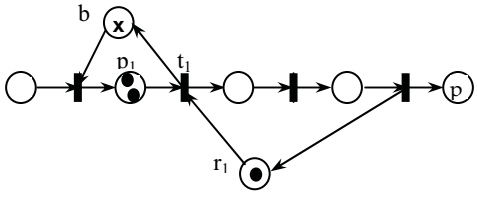
5.2.5 Handling resources and deadlocks

The work presented here assumes that, when an error occurs, all resources involved in the operation that failed and the part that was being process or manipulated become temporarily unavailable. Consider an example where two recovery actions are required to overcome an error. This could correspond to a situation of a robot dropping a part. To recover the part the part must first be found and then a command for the robot to "pick up part" must be given. Vision systems have been used for the first action of finding the part. It should be noted that during the execution of recovery actions both the resource and the part remain unavailable for other tasks. This differs from our previous work (Liu, 1993) which considered machine breakdowns in which only the machine that failed remains unavailable during the failure and repair period. The actual manipulation of a part during the failure states is considered in the logic of a workstation control agent. If the selected trajectory is an input conditioning subnet, the resources that intervened in the operation that failed remain unavailable until the operation is successfully completed. For backward and forward recovery the procedure is more complex in that all resources required to execute the operation that failed may need to be released at some point (to be determined by the recovery agent) in the recovery trajectory. Another issue is the possible occurrence of deadlocks in net augmentation. The policy adopted was to maneuver out of such deadlock states by temporarily allowing a buffer overflow. An example of maneuvering out of the deadlock situation using a Petri Net model is given in Figure 11. In the Petri net illustrated,

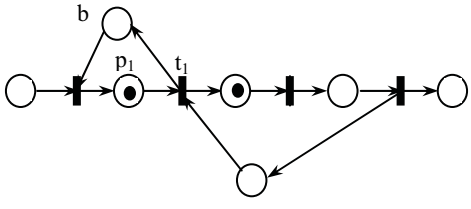
the transition tr will be allowed to fire even if no tokens are available at place b1 (i.e, the buffer b1 is full). In that case, the place p1, representing the "parts in buffer" condition, would accept a token overflow (two tokens instead of one) only for the case of tokens coming from recovery subnets. The advantage of this policy is that clears the deadlock situation in an efficient way that addtionally can be automatically generated in computer code. It should be note that if this policy is not feasible in a real system due to buffer limitations, human intervention may be required.



(a) Deadlocked net before firing t_r



(b) Firing and deletion of t_r and the corresponding arcs and places. Overflow of tokens occurs at the buffer to avoid a deadlock. X represents a negative token



(c) Firing of t1 restores the original buffer capacity

Fig. 11. Deadlock Avoidance by Allowing Temporary Buffer Overflow (Odrey and Mejia, 2005)

Another issue considered was the situation where firing t1 twice would put two tokens in place b1 and the original buffer capacity would be permanently doubled. In a Petri net this overflow condition was modeled with negative tokens. Negative tokens for Petri Nets have previouusly been proposed for automated reasoning (Murata and Yamaguchi, 1991).To compensate for an overflow situation our procedure was as follows: when a token coming from a recovery net arrives to a buffer, one token is substracted from the buffer place (in this case, the place b1 that represents the buffer availability) even though the buffer place has no available tokens. If the buffer place has no tokens available then a buffer place will contain a "negative" token representing the temporary buffer overflow. In the approach taken negative tokens indicated that a pre-condition of an action was not met but still the action was executed. The overflow is cleared when transitions, which are input to the buffer place, are fired as many times as ther are negative tokens that reside in the buffer place. The

storage buffer remains unavailable for other incoming parts from the original net until both the overflow is corrected and one slot of the buffer becomes empty. In terms of the Petri net of Figure 10, the buffer will be available again only when there is at least one token in the "buffer" place b1.

5.3 A combined neural net - Petri net approach for diagnostics

In an attempt to investigate an "intelligent" manufacturing workstation controller an approach integrating Petri net models and neural network techniques for preliminary diagnosis was undertaken. Within the context of hierarchical control, the focus was on modeling the dynamics of a flexible automated workstation with the capability of error recovery. The work-station studied had multiple machines as well as robots and was capable of performing machining or assembly operations. To fully utilize the flexibility provided of the workstation, a dynamic modeling and control scheme was developed which incorporated processing flexibility and long-term learning capability. The main objectives were (i) to model the dynamics of the workstation and (ii) to provide diagnostics and error recovery capabilities in the event of anticipated and unanticipated faults. A multi-layer structure was used to decompose complex activities into simpler activities that could be handled by a workstation controller. At the highest layer a TCPN represented generic activities of the workstation. Different color tokens served to model different types of machines, robots, parts and buffers that are involved in the system operation. This TCPN model is based on modules which model very broad workstation activities such as "move", "process" or "assemble". A processing sequence is built by linking some these modules following the process plan. Then the resources needed to execute these activities are linked. Figure 3 shows an example of the move and assemble modules. If changes are required, the designer only needs to re-assemble the activity modules.

Our goal was to provide responsive and adaptive re-actions to variation and disruption from a given process plan or assembly sequence. Specifically, three subproblems were in this research : (1) a workstation model was constructed which allowed a top-down synthesis and integration of various control functions. The proposed workstation model had several levels of abstraction which decomposes operation commands requested by a higher cell level into a sequence of coordinated processing steps. These processing steps were obtained through a hierarchical decomposition process where the corresponding resource allocations and operations synchronization problems are resolved. The motion control function is incorporated at the lowest level of the hierarchy which has adequate intelligence to deal with uncertainties in real-time, (2) a model-based monitoring scheme was developed which includes three functions : collecting necessary information for determining the current state of the actual system, checking the feasibility of performing the current set of scheduled operations, and detecting any faulty situation that might occur while performing these scheduled operations. A Petri net-based watch-dog approach was integrated with a neural network to perform these monitoring functions, and (3) an error recovery mechanism was proposed which determines feasible recovery actions, evaluated possible impacts of alternative recovery plans, and integrates a recovery plan into the workstation model (Ma, 2000; Ma & Odrey, 1996) . Our focus here is on the integration of Petri Net based models and neural network techniques for preliminary diagnostics.

Diagnostics determines the fault or faults responsible for a set of symptoms. A diagnosis may require a complete knowledge of the physical structure of the present devices and their

functionality (deep knowledge) and a short series of pre-established actions (shallow knowledge) for pre-defined faults. The diagnostics activity, as structured by Ma (2000), can be divided into two main types: (i) Preliminary diagnostics and (ii) deep reasoning. The neural network architecture for preliminary diagnostics is shown in Figure 12. Preliminary diagnostics is the first subtask of the diagnostic subfunction and is used to facilitate the diagnostic process. The approach taken here contains three different neural networks as shown in Figure 12. Neural net 1, termed NN l, generates the expected system status by converting a Petri net representation into a neural network structure for real-time control. The second neural net NN2 implements a sensor fusion and/or logical sensors concept (Henderson & Shilorat, 1984) to provide NN3 with the actual system status such that a sensory-based control system can be realized. NN3 is a multilayer feedforward neural network for classifying data obtained from NN1 and NN2 into different categories for preliminary diagnostics. Preliminary diagnostics provided a scheme to reduce efforts for further diagnostics by classifying conditions for recovery into four categories: (i) shut down the system, (ii) continue operation, (iii) call operator or (iv) invoke proper operation. The purpose of the deep reasoning module was to isolate the failure(s) and report to the error recovery module. Ma (2000) investigated a neural network model for preliminary diagnostics using an input-output technique for shallow knowledge. A Petri Net embedded in a neural network was used to classify errors. These errors were linked to a rule-based expert system containing pre-defined preliminary corrective actions (Ma and Odrey, 1996). The neural network was trained and tested with examples drawn from combinations of PN states and sensory data. Deep reasoning was not considered in Ma's work and is a subject of on-going research.

A top-down Petri net decomposition approach was performed to construct a hierarchical PN model for the given work-station example. High level Petri nets such as TCPN and TPN are included to enhance the modeling capability and the hierarchical concept provided the necessary task decomposition. The first (highest) sublevel was a timed-colored Petri net (TCPN) which is a general PN with two additional parameters: 1) a time factor to represent the operation time for each operational place, and 2) color tokens to distinguish between parts. This is decomposed into the second sublevel which is a timed Petri net (TPN) where color tokens are not required because different parts (color tokens) are modeled separately. The third decomposition (sublevel) of the model further decomposes the operations at the assembly table into detailed processing steps such as "pick up", "transport", and "place". This final decomposition allows the Petri net to be more easily analyzed.

The approach taken in this research embedded a Petri net model in a neural network structure and was termed Petri Neural Nets (PNN). The purpose of a PNN is to facilitate the process of obtaining state evolution information (the expected system status) by taking advantage of the parallel computational structure provided by neural networks and utilizing the T -gate threshold logic concept proposed by (Ramamoorthy & Huang, 1989). The state evolution of a system modeled by Petri nets can be expressed using the following matrix equation:

$$M(K+1) = M(K) + U^T(K)A, \quad K=1,2,\ldots \tag{7}$$

M(K) is a (lxm) row vector representing the system marking at the Kth stage. U(K) is a (n x 1) column vector containing exactly one nonzero entry "I" in the position corresponding to the transition to be fired at the Kth firing. The matrix A is a (nxm) transition-to-place incidence matrix. A schematic of the NN1 architecture is indicated by Figure 13.

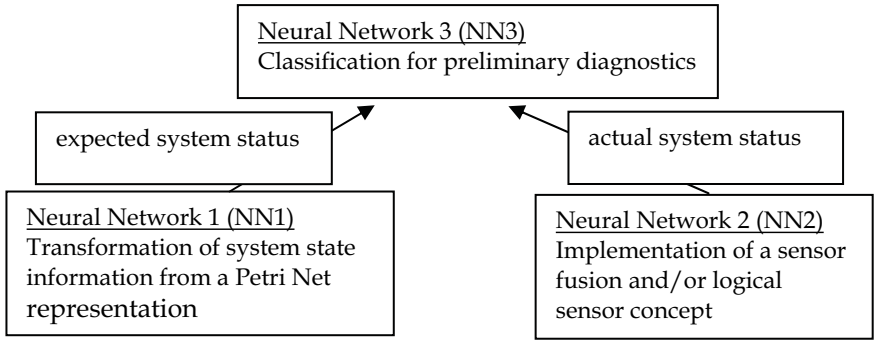


Fig. 12. Neural Network architecture for preliminary diagnosis

Based on the state equation, a three-layered PNN with an embedded T -gate threshold logic which simulated the state evolution of a general PN from M(K) to M(K+I) was developed as follows for the different layers: 1) an input vector I_k = [I_1, ...,I_m] (m = number of places) is set equal to M(K). The expected output vector O_i (i= 1,…,m) is M(K+I). The second layer of the PNN contains three vectors: (i) V_J (i=1,2 ,... , m) representing M(K), (ii) G_r (r= 1,…n) where n = number of transitions representing UT(K) which is determined by execution rules for Petri nets, and 3) H_h (h=l, …, m) which represents UT(K)A. For a decision-free PN, the execution rules can be implemented using AND T -gate threshold logic. The T -gate threshold logic is a neural network with fixed weights and can be used to implement a rule-based expert system for time-critical applications as noted by (Ramamoorthy and Huang, 1989). The weights in the PNN are hard weights and are assigned according to specified rules. Details can be found for theses weights and the output function for each layer in (Ma & Odrey, 1996).

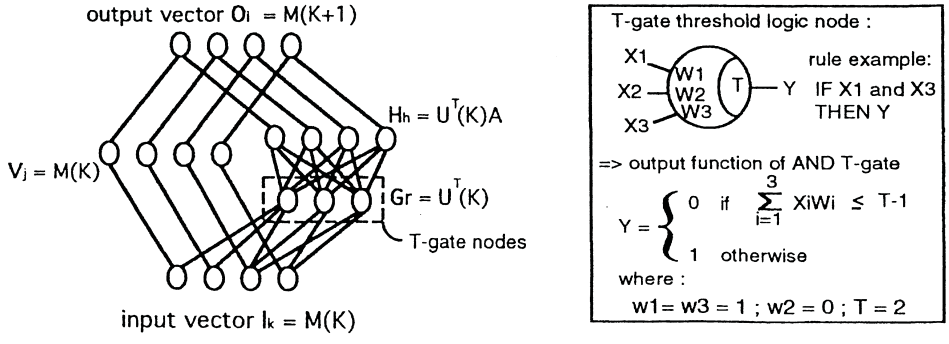


Fig. 13. NN1 Neural Network architecture incorporating T-gate threshold logic gates (Ma & Odrey, 1996)

The purpose of preliminary diagnostics was to classify operation conditions occurring in the workstation into several categories, each one associated with a preliminary action. The input vector of NN3 is portioned into two sets of nodes. The first set represents the expected system status and is obtained from the output of NNI (i.e. M(K.+I) of the corresponding sublevel-TPN model). The second set of nodes [S1, S2, ... Sn] represent categories of sensor information which are obtained from NN2. The output vector of NN3 represents the four preliminary actions: shutdown (O1), call operator (O2), continue operation (O3), and invoke further diagnostics (O4). The value of these output are either "0" representing not activated, or "1" representing activated. An outline of the system is given in Figure 13. Training and testing data are obtained using diagnostic rules based on common knowledge about the system. In general, the actua1 operation status of a system at any instant is the set of readings of all the sensor outputs. However, the actual system status information given by the sensor outputs is not sufficient for determining preliminary actions. Both the actual system status and the expected system status are required. The determination of a preliminary action for operations can thus be stated for the example of Figure 14 as follows:

IF "the expected system status" = [p1,p2.p3,p4,p5] AND "the actual system status" = [s1.s2.s3.s4]

THEN "preliminary action" = Oi (i = 1,2,3,4)

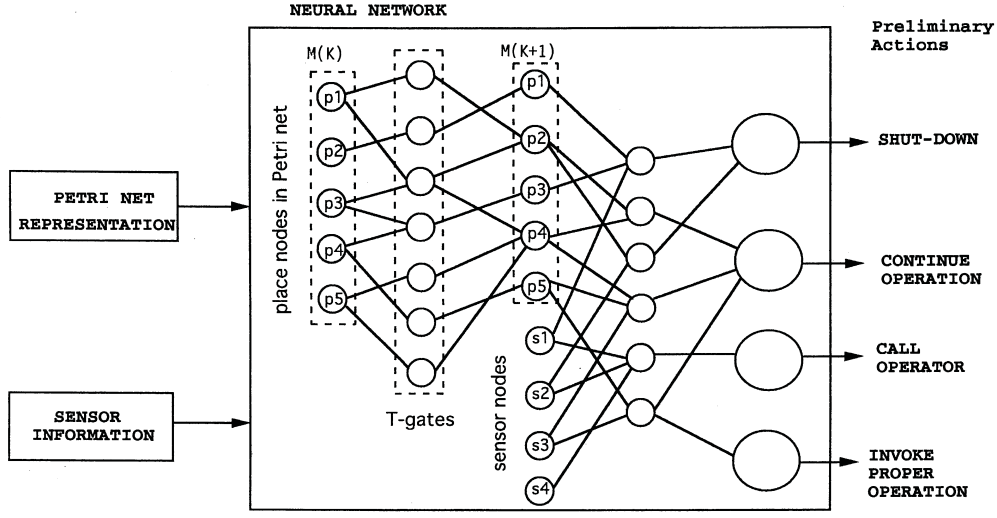


Fig. 14. Generation of preliminary actions in a neural network incorporating T-gate threshold logic

Based on a sublevel TPN model, NN1 generates different outputs corresponding to possible expected system status M(K). Different fault scenarios were used as the basis for simulation of actual system status and for generating diagnostic rules. Details of the simulation and results can be found in (Ma and Odrey, 1996). In general a neural network for preliminary diagnostics was investigated. For NN3 (classification for preliminary diagnostics) different 3-layer perceptron networks with different hidden nodes were simulated and it was found that a 19-15-4 perceptron network gave the lowest percent classification. Note that this work

did not construct the NN2 network and only simulated data was used to test the proposed neural network NN3. We plan to continue this approach which incorporates a hybrid neural – Petri net in future research.

5.3.1 Advanced diagnostics and error recovery

Preliminary diagnostics, as noted in the previous section, provides a scheme to reduce efforts for further diagnostics by classifying conditions to be diagnosed into four categories, each one associated with a preliminary action. The preliminary actions separate the diagnostic conditions which require knowledge about the physical structure of the devices and/or their functional descriptions (i.e., deep knowledge). from the conditions which need only a short series of inferences but fast responses (i.e. shallow knowledge). Shallow knowledge which usually appears in the form of direct input-output association can store patterns of predefined instructions from designers and/or experts was considered more desirable at the preliminary diagnostics stage in this research.

5.3.2 Further diagnostics

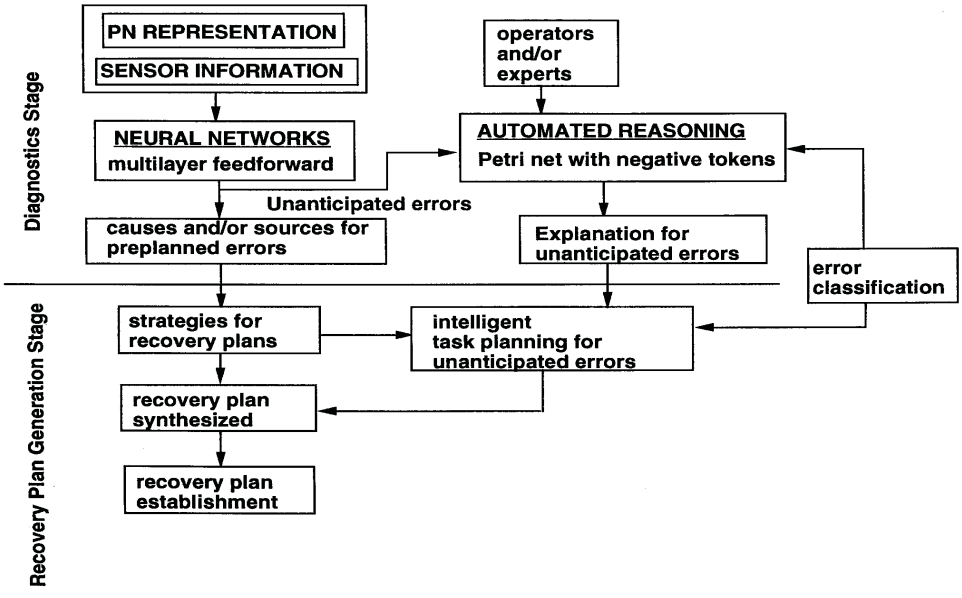


Fig. 15. A general framework for error recovery in a Petri net based system

Further (advanced) diagnostics is initiated to consider two possible situations: either a preplanned error(s) has occurred or an unanticipated error(s) has occurred. Regardless of error type, a recovery plan is needed to construct a recovery trajectory to bring the system back to a normal condition (nominal trajectory). For preplanned errors, the corresponding error causes and/ or sources can be established in a failure reason data structure. With such a database structure, one can then obtain the failure reasons associated with a particular operation. In this research, an integrated approach which utilizes both knowledge-based systems and neural networks is proposed for

unanticipated errors. Neural networks are used to provide additional information about unanticipated situations through learning. The same neural network used in the preplanned error is used to get as much information as possible about unanticipated errors. The research effort is directed toward using preplanned errors as training data and a multilayer, feedforward network as the initial test structure. A knowledge-based system then takes this information as inputs to automated processes. The modeling process is based on the feasibility of using Petri nets with negative tokens (Murata and Yamaguchi, 1990). Our current efforts focus on developing an automated reasoning technique which can draw conclusions from unknown errors in a workstation environment. To develop an automated reasoning scheme, a corresponding Petri net is established from information gathered by the neural net approach to model the reasoning. A schematic of the general framework for error recovery is given in Figure 15.

5.3.3 Error recovery strategies

After diagnostics, the workstation controller needs to generate a recovery plan to return the system back to a normal state and to continue the remaining tasks. The generation of recovery plans involves determining recovery strategies, constructing recovery activities, synthesizing a recovery sequence, and establishing a recovery plan. To determine recovery strategies, general and specific rules may be selected as constraints in the generation of recovery plans. In particular, preplanned errors and unanticipated errors usually have different sets of rules to be followed. In the case of preplanned errors, the construction of recovery activities can be easily done by recalling from computer memory. For unanticipated errors, however, an intelligent task planning system is required, and at least one feasible set of recovery activities needs to be constructed. In the approach taken recovery activities are synthesized with the planned activities to form a sequence of coordinated primitive activities. Finally, a complete recovery plan is established which includes not only the recovery actions but also other information or commands. In the research done to-date the most important issues in the generation of recovery plans was to develop an intelligent task planning system and to synthesize Petri nets corresponding to the recovery activities and to the planned activities. The purpose of an intelligent task planning system is to select and sequence processing steps that will change the current state of the system into a desired system state.

A Petri net based processing step representation to establish error recovery trajectories through a neural network based learning mechanism was undertaken. The processing steps modeled by Petri nets were categorized into two classes, namely, an action-class and a condition-class. Processing steps such as “move”, “ process”, and “assemble” that execute a task and usually have time associated with them are considered as an action-class. The condition-class processing steps represent the preconditions and/or post-condition of an action-class processing step. Examples of condition-class processing steps include “part in IB” and “part finished processing”. Every action-class processing step is followed by condition-class processing steps. Similarly, a condition-class processing step can trigger one or more action-class processing steps. Based on the relationship between action-class and condition-class processing steps, two sets of problems are defined:

P1: Action-Condition Problem (ACP), i.e. given an action-class processing step, find a (pre) condition-class processing step

P2: Condition-Action Problem (CAP), i.e. given a (post) condition-class processing step, determine an optimal action-class processing step

The recovery plan generation problem then involves solving ACP and CAP iteratively which then generates a sequence of processing steps until a desired system state is reached. When the error recovery module is initiated by the monitoring and diagnostics module, the expected system state is compare with the actual system state to obtain the discrepancy (error) of the system. If the error state is at an action-class processing step, the ACP problem is solved (through the Action Neural Network) and the result is compared with the normal trajectory to see if any of the normal state can be reached. If not, the error recovery routine continues by feeding the results from the ACP problem into the CAP problem which is solved through the Condition Neural Network. The ACP and CAP problems are invoked iteratively until a state in the normal trajectory can be reached. Similarly, if the error state is at a condition-class processing step, the CAP problem is invoked first and the results are fed into the ACP problem, if necessary.

To solve ACP and CAP problems, it was necessary to consider the interactions between action-class processing steps and condition-class processing steps. In a workstation environment, many different processing steps can be constructed. It would be difficult to consider all the interactions among all the processing steps. The basic elements for constructing a processing step, however, are limited and thus manageable. We term these individual steps as primitive elements. Our approach consisted of action-class processing steps being composed of three different elements: the action element, the object element, and the location element. For example, in the "move part A to m1 using robot 1" processing step, the action element is "move", the object elements are "part A" and "robot 1", and the location element is "m1". Similarly, the condition-class processing steps have the object element, the location element, and the status element. For example, the processing step, "part A finished at m1", has "part A" as an object element, "m1" as the location element, and the status element is "finished". Various action-class and condition –class elements can be constructed. In an industrial setting such steps could be constructed from basic Method-Time-Measurement (MTM) data already available. Each processing step is represented in terms of different elements using binary vector representations. An action-class processing step, "move part A to machine 1 with robot 1", can then be represented as an action –class vector PSA

$$PSA = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]$$

where the "1" designation refers to the primitive element considered and "0" is interpreted as an element not considered from the action-class set. Similarly, a vector PSC can be defined to represent a condition-class processing step. An example of a condition-class processing step, "part A at machine 1", could be represented by a vector PSC as follows:

$$PSC = [1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

The elements within the vector are interpreted as active or inactive. This representation has the advantages of being able to represent many combinations of actions, objects, locations, and status. In addition, the vector-based representation allows one to apply neural network techniques that provide learning capability in the generation of recovery plans for unanticipated errors. In this research, in order to capture the relationship among processing steps and to generate error recovery plans, a Boltzmann machine neural network was investigated.

5.3.4 Boltzmann machine neural network structure

The Boltzmann Machine is a particular class of neural networks that consists of a network of simple computing elements. The states of the neurons are binary, i.e. 0 and 1. The neurons in the network are connected by synapses with different (real) weights, which represent a local quantitative measure for the desirability that the two connected neurons are on. Similar to backpropagation neural networks, Boltzmann machines can be trained on test data to associate input and output values. In addition, one can use Boltzmann in optimization problems where the state of an individual neuron is iteratively adjusted to achieve minimal cost objective. The ability of doing both association and optimization makes Boltzmann machines very appealing in the application of workstation recovery plan generation. In this research, the Boltzmann machine is used at two different stages, namely a learning stage and an optimization stage. At the learning stage, the objective is to capture the relationships among various elements of the processing steps through weights adjustment. The relationships among various elements of the processing steps should be the same throughout the operations. Therefore, the learning stage is performed off-line. Once the relationships (weights) are established, the desired output is found at the second stage, on-line, through solving an optimization problem. In this research, in order to capture the relationship among process steps ant to generate error recovery plans, a Boltzman machine neural network was used. Details of this investigation are beyond the scope of this chapter and are currently being submitted for publication. Details can also be found in (Ma, 2000).

6. Conclusions

The work presented above essentially summarizes past and on-going work within the Industrial & Systems Engineering department at Lehigh University on "smart" systems. The research undertaken indicates a variable architecture and approach for such systems. Extensions to this work will incorporate stochastic implications, communications and negotiation strategies between agents, and further work on control nets and strategies. Hybrid nets such as the Petri –Neural Net are of particular interest. The techniques integrated into this work in the future will be directed toward development of robust, reconfigurable, adaptable large scale systems. Applications are currently in production and logistic systems. Other applications are being pursued.

7. Acknowledgments

The author would like to thank the students who have contributed to this work over the years. In particular, the work in this chapter is based on the work of Drs. Cheng-Sheng Liu, Christina Ma, and Gonzalo Mejia. The author would also like to thank Ms Julie Drzymalski for helping in proof reading this manuscript and providing helpful suggestions in its formulation. Her graduate dissertation is extending the concepts of this chapter to supply chains and enterprise level problems.

8. References

Albus, J. (1997). The NIST Real-time Control System (RCS): an approach to intelligent systems research. *Journal of Expert Theory in Artificial Intelligence*. Vol. 9, No 2-3, pp. 157-174.

Barad, M. & Sipper, D. (1988). Flexibility in Manufacturing Systems: Definition and Petri Net Modeling, *International Journal. of Prod. Research*, Vol. 26, No.2, pp. 237-248.

Brennan, R. (2000). Performance Comparison and Analysis of Reactive and Planning-based Control Architectures for Manufacturing. *Robotics and Computer Integrated Manufacturing*. Vol. 16 , No. 2-3, pp.191-200.

Duffie, N. Chitturi, R. Mou, J. (1988). Fault Tolerant Heterarchical Control of Heterogeneous Manufacturing System Entities. *Journal of Manufacturing Systems*. Vol. 7, No. 4. pp. 315-327.

Fielding, P. J., DiCesare, F., Goldbogen, Geof., Desrochers, A.(1987) Intelligent automated error recovery in manufacturing workstations. *Proceedings of IEEE International Symposium on Intelligent Control* 18, pp. 280-285, Philadelphia, PA, 1987, IEEE, Piscataway, NJ.

Gou, L. Luh, P. Kyoya, Y. (1998). Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. *Computers in Industry*. Vol 37, No. 3, pp. 231-231.

Henderson, T., Shilcrat,E. (1984), Logical Sensor Systems, *Journal of Robotic Systems*, Vol.1, No.2, pp. 169-193.

Hillion, H. Proth, J.M. Performance Evaluation of Job-Shop Systems Using Event Graphs (1989). *IEEE Transactions on Automatic Control*, Vol 34, No 1, pp. 3-9.

Jennings, N.R. (2000) On agent-based software engineering, *Artificial Intelligence*, Vol. 117, No. 2, pp. 277-296.

Liu C. S. (1992). Planning and Control of Flexible Manufacturing Cells with Alternative Routing Strategies. *Ph.D. Dissertation*. Department of Industrial Engineering, Lehigh University.

Liu, C, Ma, Y. Odrey, N. (1997) Hierarchical Petri Net Modeling for System Dynamics and Control of Manufacturing Systems. *Proceedings of the FAIM Conference*, pp.169-182, Middlesbrough, UK, June 1997, Begell House, NY.

Ma, Yi-Hui (2000) Flexible Manufacturing workstation with Error Recovery Capability, *Ph.D.Dissertation*, Dept. Of Industrial Engineering, Lehigh University

Ma, Yi-Hui, Odrey, Nicholas G.. (1996), On the application of Neural Networks to a Petri net –based intelligent workstation controller for manufacturing, *Proceedings of theArtificial Neural Networks in Engineering (ANNIE '96) conference*, pp. 829-836, Vol. 6, St. Louis, MO, November, 1996 ASME Press, NY.

Maturana, F. Shen, W. Norrie, D. (1999). MetaMorph: An adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, Vol. 37, No.10, pp. 2159-2173.

Mejia & Odrey(2005), An approach using Petri Nets and improved heuristic search for manufacturing systems scheduling, *Journal of Manufacturing Systems*, Vol. 2, No. 2, pp. 79-92.

Mejia, G. , Odrey, N. (2004) Real Time Control and Error Recovery of Flexible Manufacturing Workstations: An Approach Based on Petri Nets, *Proceedings of the 14th International Conference on Flexible Automation and Intelligent Manufacturing*, pp. 824-831, Toronto, CN, June, 2004, Begell House, NY.

Meystel & Albus, (2002), *Intelligent Systems: Architecture, Design , and Control*, John Wiley &Sons, Inc. , New York,

Meystel, A and Messina, E. (2000) The Challenge of Intelligent Systems, Proc. *Of 15th Int'l Sym. On Intelligent Control*, pp. 211-216, Rio Patras, Greece, July, 2000, IEEE, Piscataway, NJ.

Murata, T. (1989) Petri nets: properties, analysis, and applications. *Proc. of IEEE*. Vol.7, No. 4, pp.541-580

Odrey & Mejia (2003), A reconfigurable multi-agent system architecture for error recovery in production systems. *Robotics & Computer Integrated Manufacturing*, Vol. 19 No. 1-2, pp. 35-43.

Odrey & Mejia(2005), An augmented Petri Net approach for error recovery in manufacturing systems control, *Robotics & Computer-Integrated Manufacturing*, Vol. 21, pp. 346-354.

Odrey, N. Ma Y-H. (2001). A Multilevel, Multi-Layer Petri Net Based Approach for manufacturing Systems Control. Proceedings of the 11th International FAIM Conference. pp. 218-228, Dublin, Ireland. July 2001., Begell House, NY.

Odrey, N. Ma, Y. Intelligent Workstation Control: An Approach to Error Recovery in Manufacturing Operations. *Proceedings of the 5th International FAIM Conference,* pp. 124-141, Stuttgart, Germany, 1995, Begell House, NY.

Okino, N. (1993) A Prototype of Bionic Manufacturing Systems in Flexible Manufacturing Systems, Past, Present, Future. Publisher, J Peklenik,. Slovenia.

Sousa, P. Ramos (1999), C. A Distributed Architecture And Negotiation Protocol For Scheduling In Manufacturing Systems *Computers in Industry*. Vol. 38, No. 2, 1999, pp. 103-113.

Sun, J. Xue, D. Norrie, D (1999). An Intelligent Production System Scheduling Mechanism Considering Design and Manufacturing Constraints. *Proceedings of the Third International Conference on Industrial Automation*, pp. 2411-2414. Montreal. June, 1999

Teng, T.E. & Black, J.T., (1990), Cellular Manufacturing System Modeling: the Petri Net Approach, Journal of Manufacturing Systems., Vol.9 No.1, pp. 45-54.

Tharumarajah, A. Wells, A. J. Nemes, L. (1996) Comparison of the bionic, fractal and holonic manufacturing system concepts. International *Journal of Computer Integrated Manufacturing*. Vol. 9, No.3, pp. 217-226.

Valckernaers, P. Bonneville, F. Van Brussel, H. Bongaerts, L. Wyns, J. (1994). Results of the Holonic Control System Benchmark at KULeuven. Proceedings of Renssealer's 4th *International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT),* pp. 128-133, Troy, NY 1994, IEEE, Piscataway, NJ.

Van Brussel, H. Wyns, J. Valckernaers, H. Bongaerts, L. Peeters, P.(1998) Reference Architecture For Holonic Manufacturing Systems: PROSA. *Computers in Industry* Vol 37, pp. 255-274.

Van Brussel, H. Bongaerts, L. Wyns, J. Valckernaers, P. Van Ginderachter, T.(1999). A conceptual framework for Holonic Manufacturing: Identification of manufacturing holons. *Journal of Manufacturing Systems*, Vol. 18, No. 1, pp. 35-52.

Venkatesh, K.; Zhou, M.(1998). Object-oriented design of FMS control software based on object modeling technique diagrams and Petri nets. *Journal of Manufacturing Systems*. Vol. 17, No. 2, pp.118-136.

Wang, L. Balasubramanian, S. Norrie, D. Brennan, R. (1998). Agent-based Control System for Next Generation Manufacturing. *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*. Gaithersburg, MD., 1998, IEEE, Piscataway, NJ.

Warnecke, H.(1993), The fractal factory. Springer-Verlag, NY.

Zhou, M. DiCesare, F. (1993) *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers. USA.

15

Estimation of Mean Response Time of Multi–Agent Systems Using Petri Nets

Tomasz Babczyński and Jan Magott
Wrocław University of Technology
Poland

1. Introduction

Performance analysis of multi–agent system can be done by experiments with real system, simulation or analytic methods. Now, multi–agent technologies, e.g., (Deloach et al., 2001; JADE), are often based on Unified Modeling Lanuage (UML) (Booch et al., 1999; UML, 2007) or its modifications. The following analytical approaches: queuing network models (Kahkipuro, 1999), stochastic automata networks (Steward et al., 1995), stochastic Petri nets (King & Pooley, 1999), stochastic process algebra (Pooley, 1999), Markov chains can be used in performance evaluation of multi–agent systems.
In this chapter, an analytical approach, which is based on Petri nets, is developed. This approach is applied to performance evaluation of layered multi–agent system. These layers are associated with the following types of agents: manager, bidder, and searcher ones. Time–out mechanisms are used in communication between agents. Our method is based on approximation using Erlang distribution. Erlang distributions create the family of distributions with different number of stages. In the paper (Babczyński & Magott, 2006a), an approximation method which is based on Erlang distribution has been applied for the above layered multi–agent system. In that paper, there was no bounds for time of waiting for messages from the agents. In present chapter, time–out mechanisms are used in communication between the agents. The chapter is an extension of the paper (Babczyński & Magott, 2006b) where PERT based approach was presented. Accuracy of our approximation method is verified using simulator. This simulator has been previously used in simulation experiments with the following multi–agent systems: personalized information system (Babczyński et al., 2004a), industrial system (Babczyński et al., 2004b), system with static agents and system with mobile agent (Babczyński et al., 2005). These systems have been expressed in standard FIPA (FIPA) which the JADE technology (JADE) is complied with.
The chapter is organized as follows. In section 2, the multi–agent system is described. Then our approximation method is presented. In section 4, accuracy of our approximation method is verified by comparison with simulation results. Finally, there are conclusions.

2. Layered multi–agent system

We consider the layered multi–agent information retrieval (*MAS*) system given at Fig. 1.
The *MAS* includes: one manager type agent (*MTA*) as Fat Agent, two bidder type agents (*BTAs*) as Thin Agents, and searcher type agents (*STAs*) as Thin Agents. One *BTA* co-operates with a number of *STAs*.

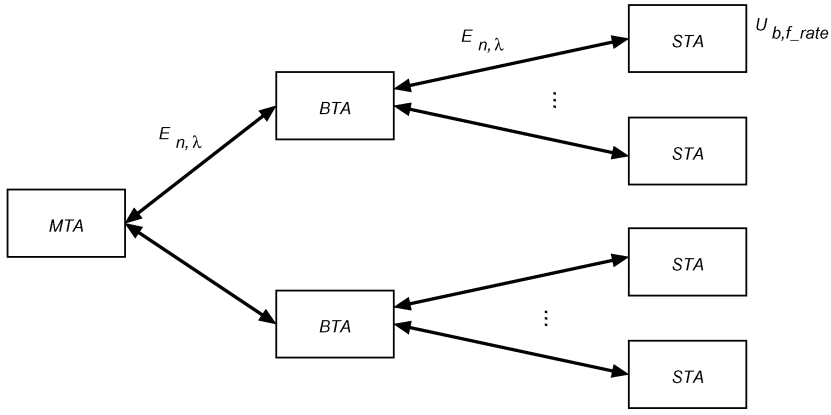


Fig. 1. Layered multi–agent information retrieval system

After receiving a request from an user, the *MTA* sends messages to the *BTAs* in order to inform them about the user's request. Then the timer of the *MTA* is started, and the *MTA* is waiting for two responses from the *BTAs*. The waiting time is limited by the termination time *tm*. Having two responses from the *BTAs*, the *MTA* prepares the response for the user. If the maximal waiting time *tm* has elapsed then the *MTA* prepares the response for the user having information received from the *BTAs* until the *tm* has elapsed. In this case, the *MTA* has the response from one *BTA* or it has no response.

After receiving a request from the *MTA*, the *BTA* sends messages to all *STAs* co–operating with this *BTA*. Then the timer of the *BTA* is started, and the *BTA* is waiting for responses from all its *STAs* but no longer than the termination time *tb*. Having responses from all its *STAs*, the *BTA* prepares the response for the *MTA*. If the maximal waiting time *tb* has elapsed then the *BTA* prepares the response for the *MTA* having information received from the *STAs* until the *tb* has elapsed. In this case, the *BTA* has less responses from the *STAs* than the number of its *STAs*.

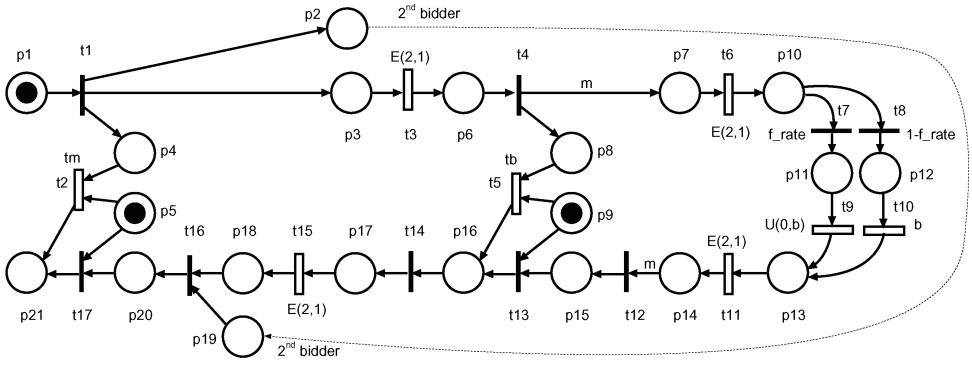


Fig. 2. Petri net model of layered multi–agent information retrieval system

The *STA* prepares the response by the Data Base (*DB*) searching. Each *STA* is associated with one *DB*. The probability of finding the response in the *DB* is denoted by *f_rate*. Time

unit is second, and it will be omitted. If there is required information in the DB, then searching time is expressed by uniform distribution over the time interval [0,*b*). Hence, the expected searching time, provided there is the required information in the *DB*, is equal to *b*/2. Searching time is equal to *b* with the probability 1-*f_rate*.
Message transmition times between the *MTA* and the *BTA*, and between the *BTA* and the *STA* are given by *n* stage Erlang distributions with parameter λ for each stage. Each stage is described by exponential distribution with this parameter. Random variable expressed by Erlang distribution is denoted by $E_{n,\lambda}$. Exponential distribution is a special case of Erlang distribution, i.e., $E_{1,\lambda}$. In examination of the accuracy of our approximation method, it will be assumed that n=2, λ=1.
In Fig. 2, rectangles represent timed transitions with non-zero firing time, while dashes represent immediate transitions with zero firing time. Immediate transitions have higher priority than timed transitions. Firing of transition t1 illustrates that the MTA sends messages to the BTAs. Presence of tokens in places p4 and p5 denotes that the timer for termination time tm has been started. If there are tokens in all input places of transition t2 during time interval of length tm, then this transition can be fired in last time instant of this interval. Firing time of transition t3 expresses transmission time of message from the MTA to a BTA. Firing of transition t4 indicates that the BTA has send m messages to m STAs. Firing time of t6 is equal to transmission time of the message from the BTA to a STA. From the other side, firing of transition t4 causes that there are tokens in all input places of transition t5. The time tb is termination time associated with the BTA. If there are tokens in all input places of transition t5 during time interval of length tb, then this transition can be fired in last time instant of this interval. If there is a token in place p10, then transitions t7, t8, respectively, are fired with probabilities f_rate, 1-f_rate, respectively. Transitions t9, t10, respectively, illustrates STA searching process in DB, provided required information is found, is not found, respectively. Transition t11 expresses transmission of message from the STA to the BTA. m tokens in place p14 shows that m messages from the STAs have been received by the BTA. There are races of transitions t5, t13 to be fired. If a token is added to place p15 earlier than the firing time tb has elapsed, then transition t13 is fired. Transition t15 is connected with transmission of message from the BTA to the MTA. Firing of transition t16 denotes that the MTA has received all messages from all BTAs. The token in place p5 is engaged in races of transitions t2, t17 to be fired. A token in place p21 indicates that the user has received a response.

3. Erlang distribution based approximation method

Now we will explain how the expected value of time of the response to the users request is approximated. First, we show what kind of operations will be considered. Then, we recall the functions used in further part of the section. Next, the method of the approximation will be shown. Finally, the approximation of the mean response time of the *MAS* will be given.

3.1 Operations on random variables

In the chapter, the approximation of the following operations will be described. In all cases we assume the independence of random variables.
Sum of *m* random variables. For *RVs* X_1 to X_m we introduce the *Sum* operation.

$$Sum(X_1,\ldots,X_m) = X_1 + \ldots + X_m \tag{1}$$

Because the *RVs* X_1 to X_m are independent, the following equations, for expected values (*E*) and variations (*Var*) of the *RVs*, are true.

$$E\ (Sum(X_1,\ldots,X_m)) = \sum_{k=1}^{m} E(X_k)$$
$$Var\ (Sum(X_1,\ldots,X_m)) = \sum_{k=1}^{m} Var(X_k) \tag{2}$$

Maximum of *m* identically distributed *RVs*. For m independent RVs X_1 to X_m we define the *Max* operation.

$$Max(X_1,\ldots,X_m) \tag{3}$$

The cumulative distribution function (*CDF*) is given by the formula:

$$F_{Max(X_1,\ldots,X_m)}(t) = (F_{X_k}(t))^m \tag{4}$$

where $F_{X_k}(t)$ is the common *CDF* of RVs X_1, …, X_m.

Cut–off by time–out event. For the RV X and time–out T we have the *Tout* operation.

$$Tout(X,T) \tag{5}$$

The *CDF* is given by the formula:

$$F_{Tout(X,T)}(t) = \begin{cases} F_X(t) & \text{for } t < T \\ 1 & \text{for } t \geq T \end{cases} \tag{6}$$

where $F_X(t)$ is the *CDF* of the RV X.

Approximation operation. Additionally, we define an *Apx* operation, which stands for the approximation of a RV by the Erlang distributed one.

$$Apx(X) = E_{n,\lambda} \tag{7}$$

where $E_{n,\lambda}$ is the Erlang distributed RV with n stages and parameter λ for each stage.
This approximation will be used for RVs resulted from the *Sum* and the *Max* operations. The details of the approximation will be shown in section 3.4.

3.2 Gamma functions

In the below described approximation, the Γ function will be used. Now we recall some equations and facts associated with this function (MathWorld). The (complete) Γ function for real value of p is defined by the following integral.

$$\Gamma(p) \equiv \int_0^\infty x^{p-1} e^{-x} dx \tag{8}$$

Two incomplete functions are also defined, the upper incomplete Γ function and the lower incomplete γ function.

$$\Gamma(p,\lambda t) \equiv \int_{\lambda t}^\infty x^{p-1} e^{-x} dx\ , \qquad \gamma(p,\lambda t) \equiv \int_0^{\lambda t} x^{p-1} e^{-x} dx \tag{9}$$

The following obvious equation is true for the all values of p and λ, t.

$$\Gamma(p) = \Gamma(p, \lambda t) + \gamma(p, \lambda t) \tag{10}$$

When the parameter p=$n \in$N, the Γ function reduces to the factorial.

$$\Gamma(n) = (n-1)! \tag{11}$$

As for the Γ function from the equation 5, the incomplete functions can also be reduced to elementary functions for the parameter p=$n \in$N.

$$\begin{aligned} \Gamma(n, \lambda t) &= (n-1)! e^{-\lambda t} \sum_{k=0}^{n-1} \frac{(\lambda t)^k}{k!} \\ \gamma(n, \lambda t) &= (n-1)! \left(1 - e^{-\lambda t} \sum_{k=0}^{n-1} \frac{(\lambda t)^k}{k!} \right) \end{aligned} \tag{12}$$

3.3 Erlang distribution

Some probability distributions of time characteristics are approximated by Erlang distribution.

The probability density function and the *CDF* of Erlang distribution with n stages and with parameter λ are given (MathWorld) by expressions:

$$f_{E_{n,k}}(t) = \frac{\lambda^n t^{n-1} e^{-\lambda t}}{(n-1)!}, \quad F_{E_{n,k}}(t) = \frac{\gamma(n, \lambda t)}{\Gamma(n)} = 1 - \sum_{k=0}^{n-1} \frac{\lambda^k t^k e^{-\lambda t}}{k!} \tag{13}$$

The random variable (*RV*) with this distribution will be denoted by $E_{n,\lambda}$. This *RV* can be interpreted as sum of n *RVs* with exponential distribution and each with parameter λ. The expected value and the variance for this *RV* are equal to $E(E_{n,\lambda}) = n/\lambda$ and $Var(E_{n,\lambda}) = n/\lambda^2$, respectively.

For the *RV* X (with any distribution), the squared coefficient of variation (*SCV*) of the X is defined by the formula:

$$SCV(X) \equiv \frac{Var(X)}{E(X)^2} \tag{14}$$

where: $E(X)$ is the expected value of X, $Var(X)$ is the variance of X.

The *SCV* for the $E_{n,\lambda}$ is equal to $SCV(E_{n,\lambda}) = 1/n$.

3.4 Two moments approximation

In the Section 3.1, the $Apx(X)$ operation was introduced. The result of the operation is the Erlang distributed random variable $E_{n,\lambda}$. In order to determine the parameters of demanded Erlang distribution, the following procedure can be used.

1. For the *RV* X under approximation, calculate the moments

   $E(X)$, $Var(X)$ and the coefficient $SCV(X)$

2. Determine the number n of stages of the Erlang distribution

   n=round(1/SCV(X))

3. Obtain the λ parameter from the equation:

$$\lambda = n / E(X)$$

While the last two steps of the procedure are common for all distributions of approximated *RVs*, the first one is different for *RVs* resulted from operations *Sum* and *Max*, introduced in the Section 3.1.

$Sum(X_1, \cdots, X_m)$ The moments are calculated from the formulae 2.

$Max(X_1, \cdots, X_m)$ In this case, the moments are calculated by numerical integrating the following formulae (Abdelkader, 2003)

$$\begin{aligned} E(M) &= \int_0^\infty \left(1 - \left(F_{X_k}(t)\right)^m\right) dt \\ Var(M) &= \left(2\int_0^\infty t\left(1 - \left(F_{X_k}(t)\right)^m\right) dt\right) - E(M)^2 \end{aligned}, \tag{15}$$

where: $F_{X_k}(t)$ is the common *CDF* of *RVs* X_1, …, X_m as in the formula 2, *M* is the shortcut for $Max(X_1, \cdots, X_m)$.

This operation will be executed on the Erlang distributed *RVs* obtained as a result of the approximation.

3.5 Example

Now, we show how to apply the approximation method to the layered multi–agent system described in the section 2.

The timed model of the system can be written using the operations defined in the section 3.1. First, we suppose that time–out mechanism is not used.

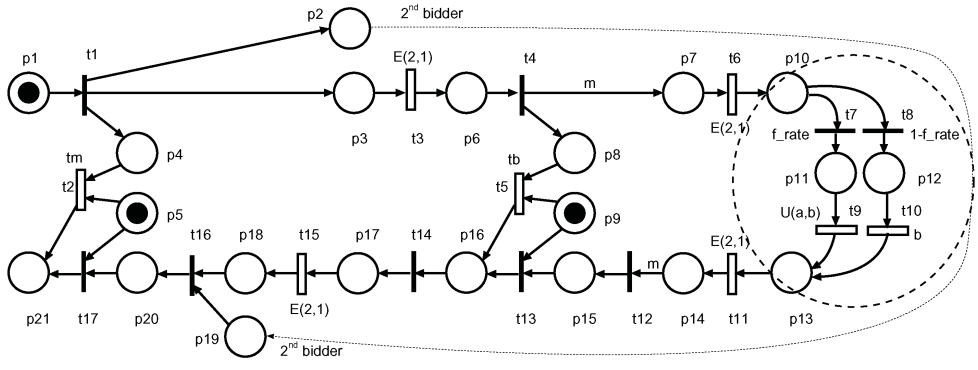


Fig. 3. Petri net of the first analysis step

Let us consider the subnet contained in dashed part of Fig. 3. It represents the *RV* of the *STA* searching time in the *DB* denoted by U_{b,f_rate}. This *RV* has the probability density function:

$$f_{U_{b,f\_rate}}(t)=\begin{cases} f\_rate\cdot 1/b & \text{for } t\in[0,b) \\ (1-f\_rate)\cdot\delta(t-b) & \text{for } t=b \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where $\delta(t\text{-}b)$ is the Dirac delta distribution in point b.
Expected value, variance, and *SCV* for this *RV* are given by the following expressions:

$$\begin{aligned} E(U_{b,f\_rate}) &= \frac{b(2-f\_rate)}{2} \\ Var(U_{b,f\_rate}) &= \frac{b^2 f\_rate(4-3f\_rate)}{12} \\ SCV(U_{b,f\_rate}) &= \frac{f\_rate(4-3f\_rate)}{3\cdot(2-f\_rate)^2} \end{aligned} \tag{17}$$

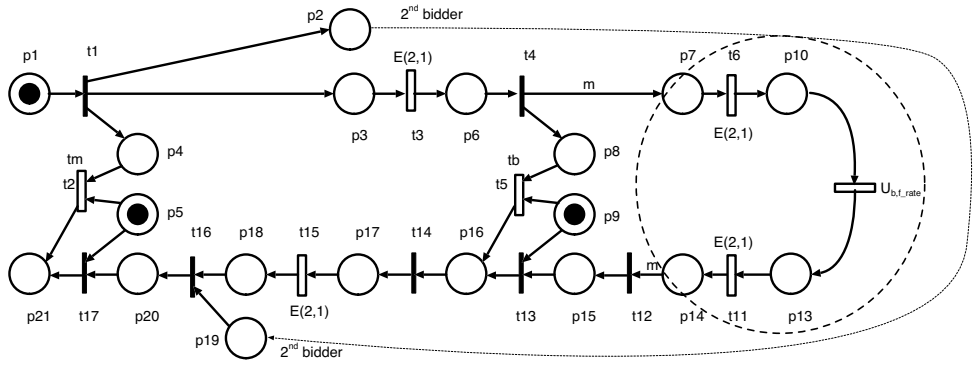


Fig. 4. Petri net of the second analysis step

Let us consider the subnet contained in dashed part of Fig. 4. It illustrates the probability distribution of the *RV X* of the length of the time interval between the time instant when the *BTA* sends the request to given *STA* and the time instant when the *BTA* receives the response from this STA. This *RV* is given by the expression:

$$X = Sum(E_{n,\lambda}, U_{b,f\_rate}, E_{n,\lambda}) \tag{18}$$

We suppose that *RVs* of the transmission times between agents and *RVs* of the searching processes in the *DBs* are independent. Hence, according to the expressions 2, the expected value, the variance, and the *SCV* for the *RV X* are expressed by the following formulae:

$$\begin{aligned} E(X) &= 2\frac{n}{\lambda} + b - \tfrac{1}{2}b\cdot f\_rate \\ Var(X) &= 2\frac{n}{\lambda^2} + \tfrac{1}{3}b^2\cdot f\_rate - \tfrac{1}{4}b^2\cdot f\_rate^2 \\ SCV(X) &= \frac{24n + \left(4f\_rate - 3f\_rate^2\right)\lambda^2 b^2}{3\left(4n + (2-f\_rate)\lambda b\right)^2} \end{aligned} \tag{19}$$

For analysed multi–agent system, the *RVs* of the transmission times between the agents are two stage Erlang distributions with the parameter λ=1 for each stage, and will be denoted by $E_{2,1}$.
The *RV X* is approximated by the *RV*:

$$E_{n_X,\lambda_X} = Apx(X) \tag{20}$$

using the procedure described in the section 3.4.

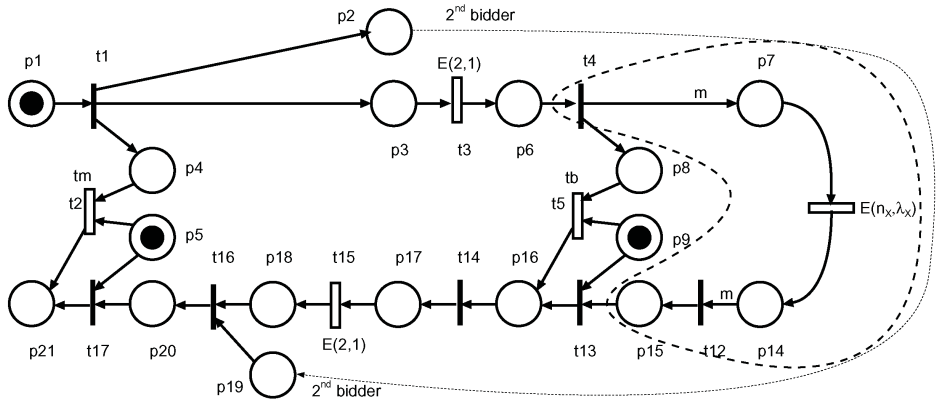


Fig. 5. Petri net of the third analysis step

Let us consider the subnet contained in dashed part of Fig. 5. It models *m STAs* associated to one *BTA*. Let $(E_{n_X,\lambda_X})_k$ be such a *RV* E_{n_X,λ_X} that approximates the length of the time interval between the time instant when the *BTA* sends the request to k^{th} *STA* and the time instant when the *BTA* receives the response from this *STA*. Equal values of mean completion times of each sequence: transmission from the BTA to the STA, searching in the DB, transmission from the STA to the BTA has been selected because this strategy usually gives the greatest error. In this case, the *RV Y* of the *BTA* waiting time for all responses from the *STAs* is

$$Y = Max((E_{n_X,\lambda_X})_1,\cdots,(E_{n_X,\lambda_X})_m)\,. \tag{21}$$

The *CDF* and the probability density function of the *RV Y* are given by the expressions:

$$\begin{aligned} F_Y(t) &= \left(\frac{\gamma(n,\lambda t)}{(n-1)!}\right)^m \\ f_Y(t) &= \frac{mt^{n-1}e^{-\lambda t}\lambda^n\left(\gamma(n,\lambda t)\right)^{m-1}}{\left((n-1)!\right)^m} \end{aligned} \tag{22}$$

The *RV Y* is now approximated by the *RV*

$$E_{n_Y,\lambda_Y} = Apx(Y) \tag{23}$$

using the procedure described in the section 3.4. The moments needed in the procedure are calculated using the formulae 15.

Let us consider the subnet contained in dashed part of Fig. 6. It represents such a situation that the *BTA* waits for the responses from the *m STAs* but not longer than for the termination time *tb*, i.e., time–out mechanism is applied. Therefore, we analyse the $RV\ E_{n_Y,\lambda_Y}$ truncated in the *tb*. This *RV* will be denoted by:

$$W = Tout(E_{n_Y,\lambda_Y}, tb) \quad . \tag{24}$$

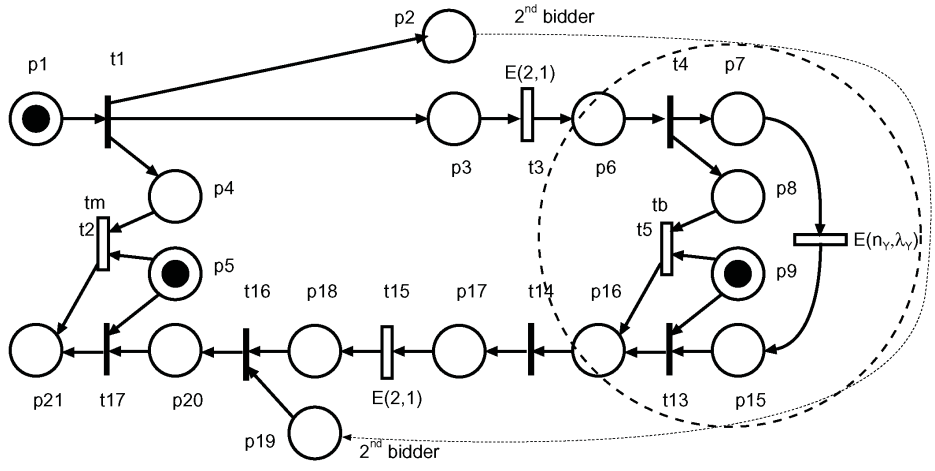


Fig. 6. Petri net of the fourth analysis step

The *CDF* and the k^{th} moment of the *RV W* are given by the expressions:

$$F_W(t) = \begin{cases} 0 & \text{for } t < 0 \\ F'_W(t) & \text{for } 0 \le t \le tb \\ 1 & \text{otherwise} \end{cases}$$

$$\text{where } F'_W(t) = \frac{\gamma(n_Y + 1, \lambda_Y t) + {\lambda_Y}^{n_Y} t^{n_Y} e^{-\lambda_Y t}}{n_Y!} \quad . \tag{25}$$

$$\mu_W^{(k)} = \frac{\gamma(n_Y + k, \lambda_Y \cdot tb)}{(n_Y - 1)! {\lambda_Y}^k} + \frac{tb^k \Gamma(n_Y, \lambda_Y \cdot tb)}{(n_Y - 1)!}$$

$$E(W) = \mu_W^{(1)}; \qquad Var(W) = \mu_W^{(2)} - (\mu_W^{(1)})^2$$

The *RV W* is not approximated.

Let us consider the subnet contained in dashed part of Fig. 7. It models the *RV* of the length of the time interval between the time instant when the *MTA* sends the request to given *BTA* and the time instant when the *MTA* receives the response from this *BTA* is denoted by the *RV*:

$$Z = Sum(E_{2,1}, W, E_{2,1}) \; . \tag{26}$$

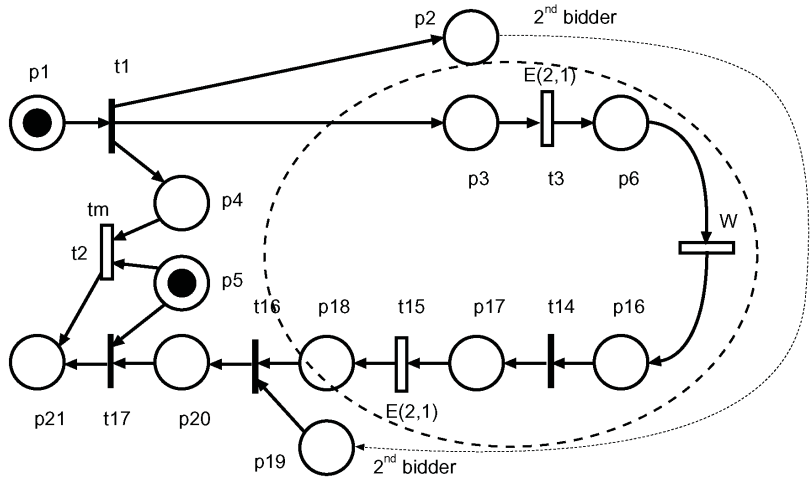


Fig. 7. Petri net of the fifth analysis step

The expected value of the time of receiving of a response by the *MTA* (or user), i.e. response time, is approximated in the similar way as the expected value of the *RVs X* and *Y* have been approximated giving the *RV*:

$$E_{n_Z,\lambda_Z} = Apx(Z)\,. \tag{27}$$

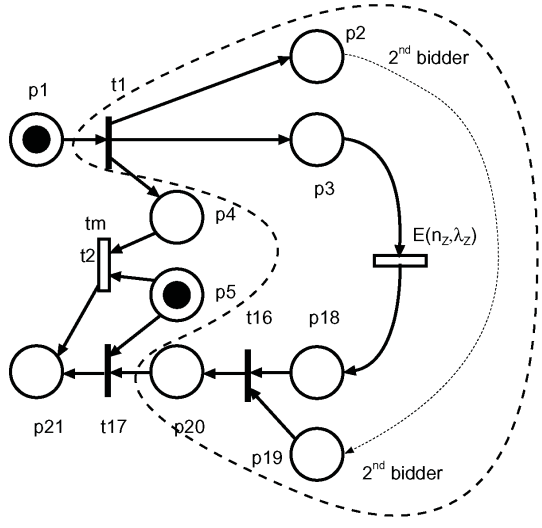


Fig. 8. Petri net of the sixth analysis step

Let us consider the subnet contained in dashed part of Fig. 8. It expresses two *BTAs* that are associated with the *MTA*. In this case, the *RV Q* of the *MTA* waiting time for all responses

from *BTAs* is:

$$Q = Max((E_{n_Z,\lambda_Z})_1,(E_{n_Z,\lambda_Z})_2)\ . \tag{28}$$

The expected value and the variance of the *RV Q* are calculated from the formulae 15. The *RV Q* is now approximated by the *RV*:

$$E_{n_Q,\lambda_Q} = Apx(Q) \tag{29}$$

using the procedure described in the section 3.4.

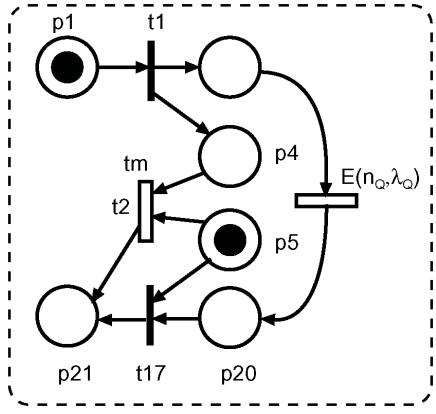


Fig. 9. Petri net of the seventh analysis step

Let us consider the subnet contained in dashed part of Fig. 9. It models that the *MTA* waits for the responses from two *BTAs* not longer than for the termination time *tm*. Therefore, we truncate the *RV* E_{n_Q,λ_Q} in the *tm*. This *RV* will be denoted by:

$$R = Tout(E_{n_Q,\lambda_Q}, tm)\ . \tag{30}$$

This *RV* is not approximated, we calculate the expected value *E(R)* in similar way as *E(W)* has been computed from equations 25.

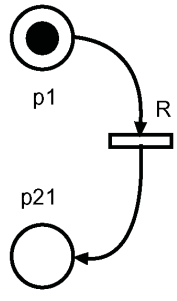


Fig. 10. The final Petri net

Summing it up, we can write our example model as:

$$\begin{aligned} X &= Sum(E_{2,1}, U_{b,f\_rate}, E_{2,1}) \\ Y &= Max(Apx(X_1), \cdots, Apx(X_m)) \\ W &= Tout(Apx(Y), tb) \\ Z &= Sum(E_{2,1}, W, E_{2,1}) \\ Q &= Max(Apx(Z_1), Apx(Z_2)) \\ R &= Tout(Apx(Q), tm) \end{aligned} \tag{31}$$

4. Accuracy of the approximation method

In order to evaluate the accuracy of the approximation method, the simulation for: the *MAS* containing *m STAs* for each *BTA*, where *m*=3,10, have been performed. For each *MAS*, the following values of *f_rate*=0.1, 0.3, 0.6, and 0.9 have been considered. The transmission time between agents is given by the *RV* $E_{2,1}$. Hence, the mean transmission time between the agents is equal to $E(E_{2,1}) = 2$, and the mean transmission time in both directions is equal to $E(tr) = 4$.

In Table 1, the percentage errors of the mean response time for: the maximal searching time in the *DB* equal to *b*=16, the termination times *tb*=20, *tm*=27 are given. First, let us suppose that time–out mechanism is not considered. The *RV* of the length of time interval between the time instant when the *BTA* sends the message to given *STA* and time interval when the *BTA* receives the response from this *STA* is $X_{16,f\_rate} = Sum(E_{2,1}, U_{16,f\_rate}, E_{2,1})$. The expected value of the *RV* U_{b,f_rate} is equal:

$$E(U_{b,f\_rate}) = f\_rate \cdot b/2 + (1 - f\_rate) \cdot b \tag{32}$$

Hence, $E(U_{16,0.9}) = 8.8$, $E(U_{16,0.1}) = 15.2$, and as a result $E(X_{16,0.9}) = 12.8$, $E(X_{16,0.1}) = 19.2$. Now let us consider the *MAS* with the time–out mechanisms. Therefore, $tb \approx E(X_{16,0.1})$.

In spite of $b/E(E_{2,1}) = 8$ and $2E(tr) < E(U_{16,0.9})$, i.e., the uniform distribution of the *RV* of the searching time is dominating the Erlang distribution of the *RV* of the transmission times, the Erlang distribution based approximation is very good (maximal error in the Table 1 is equal to 2.6%).

m \ *f_rate*	0.1	0.3	0.6	0.9
3	0.7%	-0.4%	-2.5%	-2.6%
10	1.4%	1.3%	0.7%	-0.1%

Table 1. Percentage errors of the mean response time for: the maximal searching time *b*=16, the termination times *tb*=20, *tm*=27

In Table 2, the percentage errors of the mean response time for: the maximal searching time *b*=16, the termination times *tb*=15, *tm*=20 are given. In this case: $1.3 \cdot tb \approx E(X_{16,0.1})$. The

approximation error is smaller than that for the *tb*=20 (Table 1).

m \ *f_rate*	0.1	0.3	0.6	0.9
3	0.14%	0.17%	-0.19%	-0.80%
10	0.22%	0.08%	-0.28%	0.04%

Table 2. Percentage errors of the mean response time for: the maximal searching time *b*=16, the termination times *tb*=15, *tm*=20

In Table 3, the percentage errors of the mean response time for: the maximal searching time *b*=16, the termination times *tb*=30, *tm*=40 are given. In this case: $1.5 \cdot E(X_{16,0.1}) \approx tb$. Now approximation error is clearly greater than in previous two tables.

m \ *f_rate*	0.1	0.3	0.6	0.9
3	7.3%	6.4%	4.3%	2.5%
10	11.7%	10.9%	12.1%	7.7%

Table 3. Percentage errors of the mean response time for: the maximal searching time *b*=16, the termination times *tb*=30, *tm*=40

In table 4, the percentage errors of the mean response time for: the maximal searching time *b*=32, the termination times *tb*=38, *tm*=45 are given. The expected values of analysed *RVs* are the following: $E(U_{32,0.9}) = 17.6$, $E(U_{32,0.1}) = 30.4$, and as a result $E(X_{32,0.9}) = 21.6$, $E(X_{32,0.1}) = 34.4$. Hence, $1.1 \cdot E(X_{32,0.1}) \approx tb$. Additionally, $b / E(E_{2,1}) = 16$. In this case, the uniform distribution of the *RV* of the searching time dominates the Erlang distribution of the *RV* of the transmission times stronger than for the Tables 1 and 2. The approximation is still very good.

m \ *f_rate*	0.1	0.3	0.6	0.9
3	0.5%	-0.7%	-3.1%	-2.5%
10	0.9%	0.7%	0.6%	1.0%

Table 4. Percentage errors of the mean response time for: the maximal searching time *b*=32, the termination times *tb*=38, *tm*=45

In Table 5, the percentage errors of the mean response time for: the maximal searching time *b*=32, the termination times *tb*=380, *tm*=450 are given. In this case, $b / E(E_{2,1}) = 16$. The approximation errors are much greater than previously. However, it is not realistic choice of parameters, because the termination time *tb* is more than 10 times greater than the mean

time of the $RV\ X_{32,0.1}$.

m \ *f_rate*	0.1	0.3	0.6	0.9
3	17.2%	11.9%	9.8%	7.6%
10	28.2%	22.3%	24.6%	21.0%

Table 5. Percentage errors of the mean response time for: the maximal searching time *b*=32, the termination times *tb*=380, *tm*=450

5. Conclusions

The approximation method of the mean response time for layered multi–agent system has been presented. This system has three layers of agents, namely, manager, bidder, and searcher type ones denoted by abreviations *MTA*, *BTA*, *STA*. After receiving a request from an user, the *MTA* sends the messages to the *BTAs* in order to inform them about the user request. After receiving a request from the *MTA*, the *BTA* sends the messages to all *STAs* co–operating with this *BTA*. In the communication, the time–out mechanisms are used. The *STA* prepares the response for the *BTA* by the Data Base (*DB*) searching. The probability of finding the response in the *DB* is denoted by *f_rate*. Searching time is expressed by uniform distribution over the time interval [0,*b*). Message transmission times between the *MTA* and the *BTA* and between the *BTA* and the *STA* are given by the random variables (*RVs*) of *n* stage Erlang distribution. In verification of accuracy of the method by simulation experiments, the transmission times have been expressed by the *RV* of two–stage one.
In the approximation method, the *RV* with *n* stage Erlang distribution is used. It has been obtained from the simulation, that the sum of the *RV* of the Erlang distribution (representing the transmission time) and the *RV* of searching time with uniform distribution can be approximated by the other *RV* of Erlang distribution with suitable number of stages. This is true even if the expected value of the *RV* of the uniform distribution is clearly greater than the expected value of the *RVs* of the transmission times.
Let us analyse the *RV* of the length of the time interval between the time instant when the *BTA* sends the message to given *STA* and the time instant when the *BTA* receives the response from this *STA*. Let this *RV* be denoted by *X*. If the maximal time when the *BTA* is waiting for the responses from the *STAs* (termination time) *tb* is equal or smaller than the expected value *E(X)* of the *RV X* then the approximation is very good. For the analysed cases, the maximal error is 3.1%. For the performed simulation experiments, if the *tb* is approximately equal to $1.5 \cdot E(X)$ then the maximal approximation error is about 12%. If the *tb* is about 10 times greater than the *E(X)* then, for analysed cases, the approximation error is about 28%. However, this is not realistic choice of the *tb*.
Many multi–agent systems have layered structure with the following agents: client assistant, brokers, execution agents. The presented performance approximation method can be used for finding the mean time of response on client request for this class of systems. In the future, we will try to get a better approximation using the general phase type distribution

(Bobbio et al., 2004) and hypoexponential distribution (Magott & Skudlarski, 1993) instead of the Erlang one.

6. References

Abdelkader, Y. H. (2003). Erlang distributed activity times in stochastic activity networks. *Kybernetika [Cybernetics]*, Vol. 39, No. 3, 2003, 347-358.

Babczyński, T.; Kruczkiewicz, Z. & Magott, J. (2004a). Performance evaluation of multiagent personalized information system. *Proceedings of the 7th Int. Conf. Artificial Intelligence and Soft Computing - ICAISC*, Zakopane, 2004, LNCS/LNAI, Springer-Verlag, Vol. 3070, 810-815.

Babczyński, T.; Kruczkiewicz, Z. & Magott, J. (2004b). Performance analysis of multiagent industrial system. *Proceedings of the 8th Int. Workshop Cooperative Information Agents - CIA*, Erfurth, 2004, LNCS/LNAI, Springer-Verlag, Vol. 3191, 242-256.

Babczyński, T.; Kruczkiewicz, Z. & Magott, J. (2005). Performance comparison of multiagent systems. *Proceedings of the Central and Eastern European Conference on Multiagent Systems - CEEMAS*, 2005, LNCS/LNAI, Springer-Verlag, Vol. 3690, 612-615.

Babczyński, T. & Magott, J. (2006a). PERT based approach to performance analysis of multi–agent systems. *Proc. of Int. Conference on Artificial Intelligence and Soft Computing ICAISC*, Zakopane, 2006, LNCS/LNAI, Springer-Verlag, Vol. 4029, 1040-1049.

Babczyński, T. & Magott, J. (2006b). Estimation of mean response time of multi–agent systems. *Software Engineering Techniques: Design for Quality*, K. Sacha (ed.), 2006, Springer-Verlag, vol. 227, 109-113.

Bobbio, A.; Horvath, A. & Telek, M. (2004). The scale factor: a new degree of freedom in phase-type approximation. *Performance Evaluation*, Elsevier, Vol. 56, No. 1-4, 2004, 121-144.

Booch, G.; Rumbaugh, J. & Jacobson, I. (1999). *The Unified Modeling Language, User Guide*, Addison Wesley Longman, 1999.

Deloach, S.A.; Wood, M.F. & Sparkman, C.H. (2001). Multiagents systems engineering, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, 2001, 231-258.

FIPA. Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/

JADE. http://jade.tilab.com/

Kahkipuro, P. (1999). UML based performance modelling framework for object-oriented distributed systems, *Proceedings of the Unified Modeling Language: Beyond the Standard*, 1999, LNCS, Springer-Verlag, Vol. 1723.

King, P. & Pooley, R. (1999). Using UML to derive stochastic Petri net models. *Proc. of the 15th Annual UK Performance Engineering Workshop*, University of Bristol, 1999, 45-56.

Magott, J. & Skudlarski, K. (1993). Estimating the mean completion time of PERT networks with exponentially distributed durations of activities. *European Journal of Operational Research*, Vol. 71, 1993, 70-79.

MathWorld, Wolfram Research, Inc.,
http://mathworld.wolfram.com/ErlangDistribution.html,
http://mathworld.wolfram.com/topics/GammaFunctions.html.

Pooley, R. (1999). Using UML to derive stochastic process algebra models. *Proceedings of the*

15th Annual UK Performance Engineering Workshop, University of Bristol, 1999, 23-33.

Stewart, W. J.; Atif, K. & Plateau, B. (1995). The numerical solution of stochastic automata networks, *European Journal of Operation Research*, Vol. 86, No 3, 1995, 503-525

UML (2007). Unified Modeling Language v.2.1, *OMG specification*, 2007 http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

16

Diagnosis of Discrete Event Systems with Petri Nets

Dimitri Lefebvre
GREAH – University Le Havre
France

1. Introduction

Modern technological processes include complex and large scale systems, where faults in a single component have major effects on the availability and performances of the system as a whole. For example manufacturing systems consists of many different machines, robots and transportation tools all of which have to correctly satisfy their purpose in order to ensure and fulfil global objectives. In this context, a failure is any event that changes the behaviour of the system such that it does no longer satisfy its purpose. Failure events lead to fault states (Rausand et al., 2004). Faults can be due to internal events as to external ones, and are often classified into three subclasses : plant faults that change the dynamical input – output properties of the system, sensor faults that result in substantial errors during sensors reading, and actuator faults when the influence of the controller to the plant is disturbed (Blanke et al., 2003).

In order to limit the effects of the faults on the system, diagnosis is used to detect and isolate the failures. Diagnosis is often associated with control reconfiguration, that adapts the controller to the faulty situation such that it continues to satisfy its goal. Fault diagnosis and controller reconfiguration are carried out by supervision systems. This chapter only consider problems related to the diagnosis of systems. Diagnosis includes distinct stages:

1. The fault detection decides whether or not a failure event has occurred. This stage also concerns the determination of the time at which the failure occurs.
2. The fault isolation find the component that is faulty.
3. The fault identification identifies the fault and estimates also its magnitude.

Diagnosis is usually discussed according to the model type used, with component based analysis that uses architectural and structure graph models, with continuous variables systems described by differential or difference equations and transfer functions, with discrete event systems represented by automata or Petri nets and with hybrid dynamical systems that combine continuous and discrete event behaviours (Blanke et al., 2003). Component based methods uses qualitative methods (Rausand et al., 2004) as failure modes and effect analysis (Blanke, 1996) and bi-partite graphs to investigate the redundancies included in the set of constraints and measurements for diagnosis purposes (Cordier et al., 2000; Patton et al., 1999). Fault diagnosis of continuous variables systems is usually based on residual generation and evaluation with parity space approaches or observation,

identification and parameters estimation techniques (Gertler, 1998; Patton et al., 1989). The behaviour of discrete event dynamical systems (DES) is described by sequences of input and output events. In contrast to the continuous systems only abrupt changes of the signal values are considered with DES. In that case, the state of the art is different in comparison with continuous approaches and only few results are available for diagnosis. The problem has been originally investigated with observation methods for automata (Sampath et al., 1995) developed in connection with the supervisory control theory (Ramadge et al., 1987).
This chapter focus on diagnosis of DES modelled with Petri nets (PN) where failures are represented with some particular transitions. The problem is to detect and isolate the firing of the failure transitions in a given firing sequence. The firings of the failure transitions are assumed to be unobserveable and must be estimated according to complete or partial marking measurements that are eventually disturbed by measurement errors. Several problems are related : firing sequences estimation, sensor selection, delay from failure event to detection, and also diagnosers complexity. Let us notice that this study is limited to the methods that represent the faulty behaviours according to the firing of failure transitions and that assume that the state (i.e. the marking vector) of the system is partially or totally measurable. In a alternative way, faults can be also considered as forbidden states. In that case, the observation of the state has been investigated in order to design controllers with forbidden marking specifications (Giua et al., 2002). Asynchronous diagnosis by means of PN unfolding techniques and hidden state history reconstruction obtained from alarm observations was also investigated (Benvenist et al., 2003). These approaches are not developped in this chapter.
The chapter is divided into six sections. Section two states the problem and introduces the notations. Section three is about state space methods that are based on a partial expansion of the reachability graph of the PN under consideration. Section four concerns structural methods that investigate the causality relationships characterized by incidence matrix. Section five is about algebraic methods inspired from coding theory in finite fields of integer numbers. The section six sums up the results and is a tentative of synthesis of the different approaches.

2. Problem statement, motivations and notations

A dynamical system with input u and output y is subject to some faults f. Basically, the diagnosis problem is to find the fault f from a given sequence of input – output couples (U, Y) with:

$$U = (u(0), u(1),\ldots,u(k))$$
$$Y = (y(0), y(1),\ldots,y(k)) \quad (1)$$

where k stands for time $t = k.\Delta t$, and Δt represents the sampling period of sensors. In the next Δt will be omitted and time t will be referred as k as long as there is no ambiguity. It is commonly assumed that no inspection of the process is possible. As a consequence the diagnosis is only based on available measurement data. Moreover the diagnosis problem is usually considered under real time constraints. As long as DES are considered the signals are not real-valued but belong to a discrete value set.

The motivations for the diagnosis of DES is obvious as long as DES occur naturally in the engineering practice. Many actuators like switches, valves and so on, only jump between discrete states. Binary signals are mainly used with numerical systems and logical values "true" and "false" are often used as input and output signals. Alarm sensors that indicate that a physical quantity exceeds a prescribed bound are typical systems with only two logical states. Moreover, in several systems also the internal state is discrete valued. As an example, robot encoders are discrete valued even if the number of discrete state is large enough to produce smooth trajectories. At last, one must keep in mind that a given dynamical system can always be considered as a DES system or as a continuous variable system according to the purpose of the investigation. As long as supervision problems are considered, a rather broad view on the system behaviour can be adopted that is based on discrete signals. On the contrary, if signals have to remain in a narrow tolerance band, the following approaches do no longer fit and one has to adopt a continuous point of view (Blanke et al., 2003).

2.1 Ordinary Petri nets

An ordinary PN with n places and q transitions is defined as < P, T, Pre, Post > where P = $\{P_i\}$ is a non-empty finite set of n places, T = $\{T_j\}$ is a non-empty finite set of q transitions, such that $P \cap T = \varnothing$. Pre: $P \times T \rightarrow \{0, 1\}$ is the pre-incidence application and $W_{PR} = (w^{PR}_{ij}) \in \{0, 1\}^{n \times q}$ with w^{PR}_{ij} = Pre (P_i, T_j) is the pre-incidence matrix. Post: $P \times T \rightarrow \{0, 1\}$ is the post-incidence application and $W_{PO} = (w^{PO}_{ij}) \in \{0, 1\}^{n \times q}$ with w^{PO}_{ij} = Post (P_i, T_j) is the post-incidence matrix. The PN incidence matrix W is defined as $W = W_{PO} - W_{PR} \in Z_3^{n \times q}$ with $Z_3 \in \{-1, 0, 1\}$ and w_i stands for the i^{th} column of W (Askin et al., 1993; Cassandras et al., 1999; David et al., 1992). $M = (m_i) \in (Z^+)^n$ is defined as the marking vector and $M_I \in (Z^+)^n$ as the initial marking vector, with Z^+ the set of non negative integer numbers. A firing sequence $\sigma = T_i.T_j\ldots T_k$ is defined as an ordered series of transitions that are successively fired from marking M to marking M' (i.e. M [σ > M') such that equation (2) is satisfied:

$$\sigma : M \xrightarrow{T_i} M_1 \xrightarrow{T_j} M_2 \rightarrow \cdots \xrightarrow{T_k} M' \tag{2}$$

A sequence σ can be represented by its characteristic vector (i.e. Parikh vector) $X = (x_j) \in (Z^+)^q$ where x_j stands for the number of times T_j has occurred in sequence σ (David et al., 1992). Marking M' resulting from marking M with the execution of sequence σ is given by (3):

$$\Delta M = M' - M = W.X \tag{3}$$

The reachability graph R(PN, M_I) is the set of markings M such that a firing sequence σ exists from M_I to M. A sequence σ is said to be executable for marking M_I if there exists a couple of markings (M, M') $\in$ R(PN, M_I) such that M [σ > M'.

2.2 Problem statement and notations

The objective of diagnosis problem is to identify the occurrence and type of failure events, based on observable traces generated by the system. For this purpose, let us define $\Delta_F = \{F_k\}$ the set of K distinct faults that may affect the system. A label $L \in \Delta = \{N\} \cup \Delta_F$ is associated

to each transition. As a consequence $T = T_F \cup T_N$ with T_F the set of "failure" transitions and T_N the set of "normal" transitions. The firing of transitions is usually unobservable. $L = N$ is interpreted as a "normal" behavior, and $L = F_k$ means that fault F_k has occurred. Starting from an initial state, the system may evolve according to a "normal" behavior by firing "normal" transitions or according to a faulty behavior by firing a sequence with one or several "failure" transitions.

Let us define $\theta = \{\theta_k\} \subset T^b$ be a list of b groups of fault transitions $\theta_k \subset T$ (or eventually single failure transitions). We define $B(\theta) = (b_{kj}) \in \{0, 1\}^{b \times q}$ such that $b_{kj} = 1$ if $T_j \in \theta_k$, else $b_{kj} = 0$. Let us also consider $X_\theta = B(\theta).X \in (Z^+)^b$ the firing vector to be estimated. In other words, the k^{th} row of matrix $B(\theta)$ characterizes θ_k, and the sum of firing occurrences in the k^{th} subset of transitions (i.e. the k^{th} entry of $X(\theta)$) has to be estimated from the measurement of the observable markings. To define a list θ of transitions subsets is interesting in case of non discernable faults. When the faults $\{F_k\}_{k = 1,...K}$ must be detected and located, then the list $\theta = \{\{T_{F1}\},\ldots,\{T_{FK}\}\}$ with K singletons $\{T_{F1}\},\ldots,\{T_{Fk}\}$ is used. When the faults $\{F_k\}_{k = 1,...K}$ must be detected but not isolated (i.e non discernable faults) $\theta = \{T_{F1},\ldots,T_{FK}\}$ with a single subset $\{T_{F1},\ldots,T_{FK}\}$ is defined.

The set P is also divided into the set $P_O = \{P'_i\}$ of c observable places and the set P_U of n - c unobservable ones: $P = P_O \cup P_U$. Vector $M_O \in (Z^+)^c$ is defined as $M_O = C(P_O).M$ with $C(P_O) = (c_{ij}) \in \{0, 1\}^{c \times n}$, such that $c_{ij} = 1$ if $P_j \in P_O$ and $P_j = P'_i$, else $c_{ij} = 0$. Only the marking M_O of the observable places is assumed to be measured. Let us also define $W_O = C(P_O).W \in (Z_3)^{c \times q}$, $w_O(j)$ as the j^{th} column of matrix W_O, and ΔM_O according to (4):

$$\Delta M_O = C(P_O).W.X = W_O.X \tag{4}$$

Petri nets are asynchronous models. As a consequence, two distinct transitions are never simultaneously fired and the following basic assumption can be considered: there always exists a marking measurement between two consecutive firings in a given firing sequence.

The preceding hypothesis is necessary because the firing of a transition will be undetectable if it does not have any observable influence on the marking variation. For example, the marking of the cycle $\{P_2, T_3, P_3, T_4\}$ in PN1 (figure 1) is not modified if there is no intermediate observation for the sequence of firings $\sigma = T_3.T_4$. Moreover the marking of a given place is not modified if a transition in the preset and another one in the post – set are both fired between two consecutive observations. For example, the marking of place P_1 in PN1 remains unchanged after the execution of sequence $\sigma = T_2.T_1$. According to the preceding hypothesis, the firing sequences that are considered in the following can always be separated into sub-sequences of size 1 : $X \in \{0, 1\}^q$, and $||X|| \leq 1$.

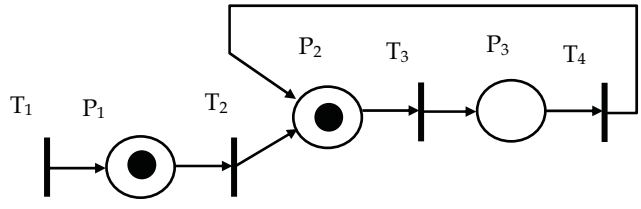


Fig. 1. Example PN1 of Petri net with cycles

3. State space methods for the diagnosis of DES

3.1 Partial expansion of reachability graph and indetermined cycles

Fault diagnosis based on state space approach and on partial expansion of the reachability graph was first formulated with automata (Sampath et al., 1995). Sampath et al. introduce the study of indeterminate cycles in automata and state that a langage is diagnosable if and only if the diagnoser satisfies the following condition : there is no F_k - failure indeterminate cycle for all failure types.

The investigation of indeterminate cycles was then extended to PN with finite reachability graph (Ushio et al., 1998). The considered PN are live (i.e. for any $T_j \in T$, and for all $M \in R(PN, M_I)$ there exists a sequence σ executable from M that includes transition T_j) and safe (i.e. for all $M \in R(PN, M_I)$, $M \in \{0, 1\}^n$) with some places that are observable and other not. Transitions are usually assumed to be unobservable. The diagnosability of the system is based on the study of indetermined cycles included in the observable part of the labelled reachability graph $R(PN, T_F, M_I, P_O)$ (Ushio et al., 1998). A cycle is called "determined" if it contains at least one observable state that results with no ambiguity from a normal firing sequence, or from a F_k - failure firing sequence (i.e. a firing sequence that contains a F_k - failure transition). Characterisation of the cycles is obtained according to label propagation and range functions that tell us how to assign the fault labels and how to estimate all the next possibly diagnoser states from an initial state. Starting from an observable initial marking, the diagnoser detects and isolates a failure transition in a given firing sequence from measurement of the successive observable states visited by the system.

The notion of diagnosability is defined as the inherent property of the system that when a failure occurred, we can always infer its type, no matter how the system evolves after the failure. The resulting diagnosers are "delayed" (i.e. multi-steps diagnoser) in the sense that the occurrence of intermediate events may be necessary to detect and isolate the faults. The number of intermediate events is upper bounded according to the maximal size of the determined cycles. In (Chung et al., 2003) some transitions are assumed to be observable in order to increase the database used by the diagnoser. An algorithm, based on linear programming, of polynomial complexity in the worst case for computing a sufficient condition of diagnosability has been also proposed (Wen et al., 2005).

Let us consider the Petri net named PN2 in figure 2 as an example. All transitions are supposed to be unobservable. The transition T_1 represents a failure event F. Other transitions are assumed to represent normal events.

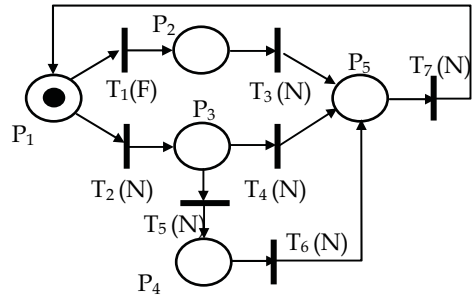


Fig. 2. Example PN2 of Petri net

If the set of observable places is given by $P_{O1} = \{P_1, P_4, P_5\}$, the observable part of the labelled reachability graph R(PN2, $\{T_1\}$, $(1, 0, 0, 0, 0)^T$, P_{O1}) is worked out as in figure 3a. This diagnoser has an indetermined cycle so the system is not diagnosable (figure 3a, on the left). If $P_{O2} = \{P_1, P_3\}$, the observable part of the labelled reachability graph R(PN2, $\{T_1\}$, $(1, 0, 0, 0, 0)^T$, P_{O2}) is worked out as in figure 3b. This diagnoser has no indetermined cycle so the system is diagnosable.

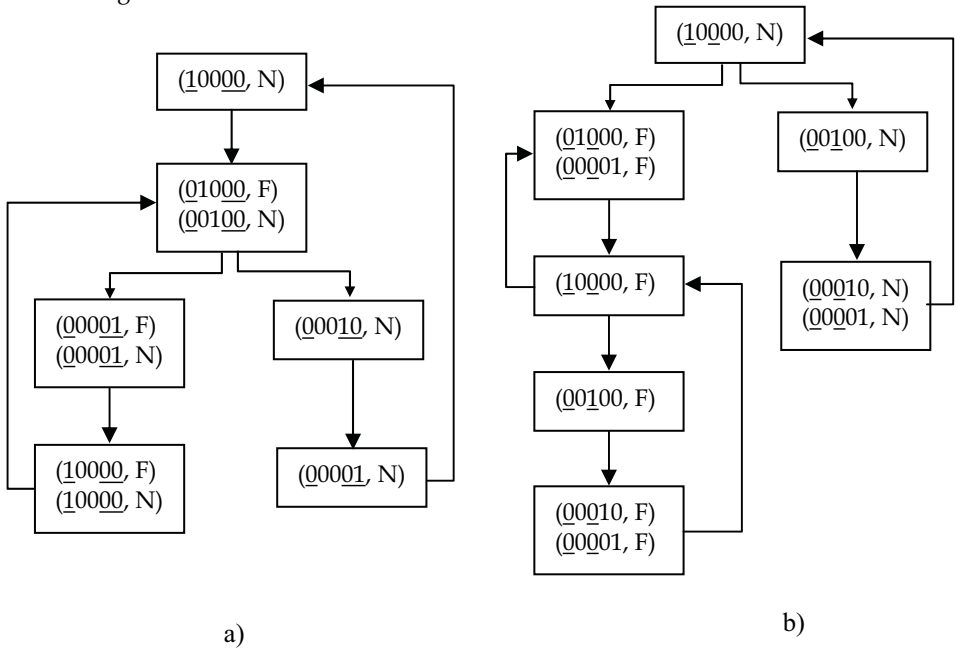


Fig. 3. Two partial expansions of the reachability graph for PN2

a) R(PN2, $\{T_1\}$, $(1, 0, 0, 0, 0)^T$, P_{O1}) ; b) R(PN2, $\{T_1\}$, $(1, 0, 0, 0, 0)^T$, P_{O2})

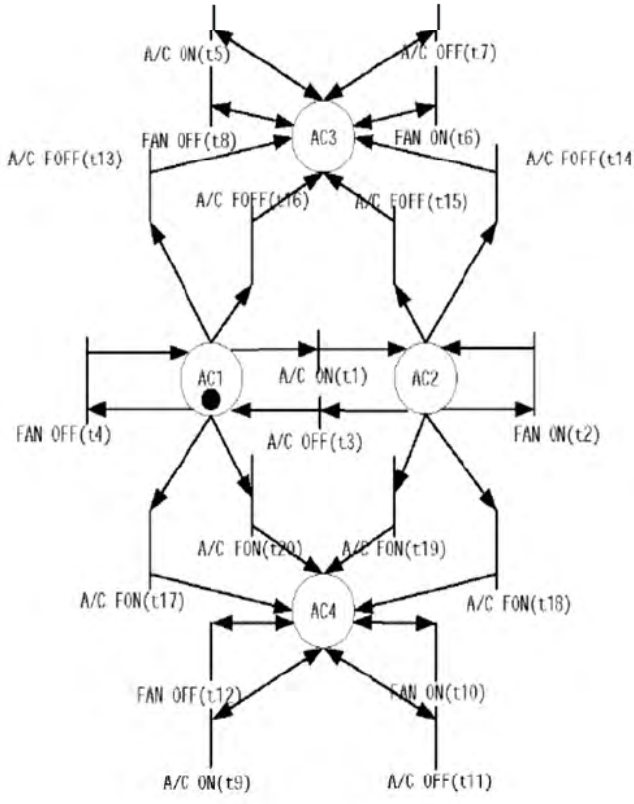
As a conclusion, let us notice that the preceding method is efficient to evaluate the diagnosability of a system but not suitable to design diagnosers. The reason is that the partial expansion of the reachability graph must be worked out for all diagnoser candidates. Such a computation is time consuming so that it cannot be adapted for sensor selection problems in case of large scale systems.

3.2 Application

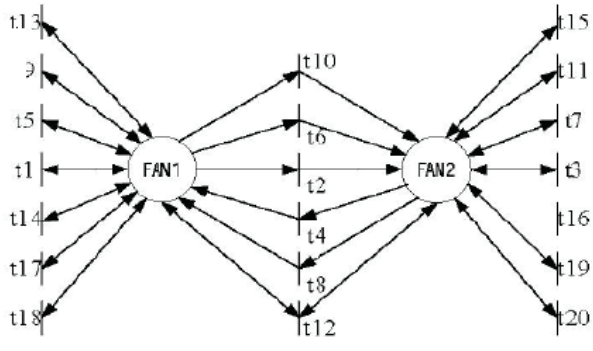
State space method have been used to state the diagnosability of an automatic temperature control system (ATC) for automobile applications (Wen et al., 2005). The PN models of ATC has 3 components (figure 4a-b-c):

a) The pump model has four unobservable states. The places ACI and AC2 stand for pump off and pump on respectively. The places AC3 and AC4 stand for pump failed off and pump failed on respectively.
b) The fan model has two unobservable states : FAN1 and FAN2 stand for fan off and fan on respectively.
c) The controller has four observable states and four events. The state Cl represents both the pump and fan are off. State C2 represents that the pump turns on first, while the fan

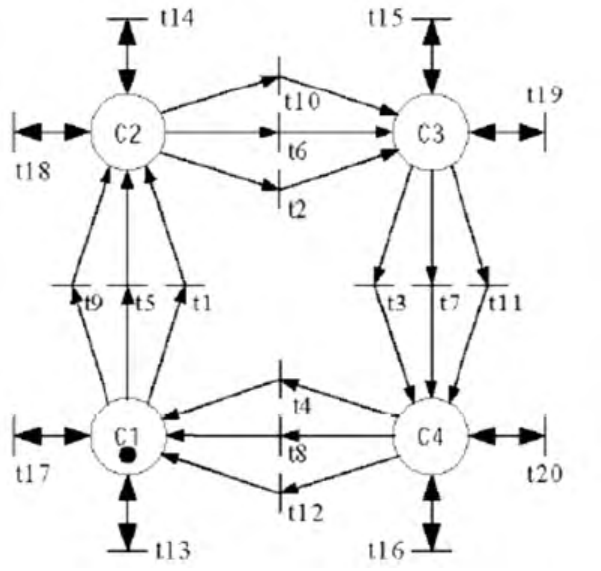
is in off. State C3 represents that the pump turns on, and the fan turns on. State C4 represents that the pump turns off first, while the fan is still working.



a) Pump



b) Fan



c) Controller

Fig. 4. PN3 model of an automatic temperature control system (Wen et al., 2005)

Transition	Event Type (Fail type)	Sensor Map
t_1	A/C ON (N)	H to H
t_2	Fan ON (N)	H to L
t_3	A/C OFF (N)	L to H
t_4	Fan OFF (N)	H to H
t_5	A/C ON (F1)	H to H
t_6	Fan ON (F1)	H to H
t_7	A/C OFF (F1)	H to H
t_8	Fan OFF (F1)	H to H
t_9	A/C ON (F2)	H to H
t_{10}	Fan ON (F2)	H to L
t_{11}	A/C OFF (F2)	L to L
t_{12}	Fan OFF (F2)	L to H
t_{13}	A/C FOFF (F1)	H to H
t_{14}	A/C FOFF (F1)	H to H
t_{15}	A/C FOFF (F1)	L to H
t_{16}	A/C FOFF (F1)	H to H
t_{17}	A/C FON (F2)	H to H
t_{18}	A/C FON (F2)	H to H
t_{19}	A/C FON (F2)	L to L
t_{20}	A/C FON (F2)	H to L

Table 1. Transitions and sensor map of the ATC (Wen et al., 2005)

There are two failure types. Failure types F1 and F2 stand for pump fails off and pump fails on respectively. It is assumed that the system has one temperature sensor. The set of outputs is L = {low} and H={high} according to the temperature in the cabin of the vehicle. The meaning of the transitions and sensor map are listed in table 1. For example, S (H to H) means that the reading of the cabin sensor changes from High to High. The study of the indetermined cycles in observable part of reachability graph and the investigation of the transitions (events) with the same observable projection (for example T_1, T_5, and T_9 represent the same observable projection $\{e_1\}$ where $\{e_1\}$ depicts that the controller state is "pump on" and it's sensor reading changes from High to High) is useful to state that this system is diagnosable.

4. Diagnosis based on structural approaches

4.1 Event detectability

Another diagnosis approach for DES has been developed according to event detectability of interpreted PN (Alcaraz-Mejia et al., 2003; Ramirez-Trevino et al., 2004). An interpreted PN is event detectable when any pair of transitions can be distinguished from each other by the observation of the input - output symbols of the interpreted PN (inputs are defined according to the events associated with the transitions and outputs are defined according to the measurements of the observable markings). Preliminary results have been obtained according to the additive independence of columns of the output matrix (Ichikawa et al., 1988). A characterization of event detectability has been established as a consequence, when all columns of matrix $W_O = C(P_O).W \in (Z_3)^{c \times q}$ are not zero and different from each other (Alcaraz-Mejia et al., 2003). Input - output diagnosability in finite number of steps has been derived as a consequence. An interpreted PN is input - output diagnosable in r steps if any marking M resulting immediately from the firing of a fault transition is distinguishable from any other marking M' by firing any sequence with r transitions (Alcaraz-Mejia et al., 2003; Ramirez-Trevino et al., 2004). Several structural characterizations of input - output diagnosability have been provided: necessary and sufficient conditions related to input - output relationships between places, sufficient conditions when the normal behaviour of the interpreted PN is event detectable (Alcaraz-Mejia et al., 2003; Ramirez-Trevino et al., 2004; Ramirez-Trevino et al., 2007).

In order to illustrate event diagnosability, let us consider again PN2 in figure 2. On one hand, if the set of observable places is given by $P_{O1} = \{P_1, P_4, P_5\}$, event detectability is worked out according to matrix W_{O1}:

$$C(P_{O1}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad W_{O_1} = C(P_{O1}).W = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & -1 \end{pmatrix} \tag{5}$$

System (5) is not event detectable because columns 1 and 2, and also columns 3 and 4 of matrix W_{O1} are identical.

On the other hand, if the set of observable places is given by $P_{O2} = \{P_1, P_3\}$, event detectability is worked out according to matrix W_{O2}:

$$C(P_{O2}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad W_{O2} = C(P_{O2}).W = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 \end{pmatrix} \tag{6}$$

However system (6) is diagnosable for fault F_1, it is not event detectable because columns 3 and 6 are zero and columns 4 and 5 are identical. As a consequence, input - output diagnosers cannot be derived.

In (Aramburo-Lizarraga et al., 2005) the condition of event detectability is relaxed over parts of the model where the faults are not expected; thus the diagnoser handles a reduced model. Moreover, a method for splitting the global model into communicating modules is proposed that leads to the design of a set of distributed diagnosers. A framework concerning DES diagnosis based on PN and event detectability approach can also be found in (Ramirez-Trevino et al., 2007) where the authors introduce a bottom-up modelling methodology that avoids tuning phases and state combinatory found in finite state automata approaches.

4.2 Minimal sets of observable places for single step diagnosis

Fault diagnosis is strongly related to the problem of sensor selection that leads to the determination of minimal sets (for inclusion) of observable places in order to detect and identify the firing of some particular "failure" transitions. In this context, places are assumed to have a physical meaning so that direct relationships exist between places, state variables and sensors. The problem is to decide the number and location of the places to be observed (i.e. the state variables to be measured) in order to estimate the firings of some transitions (i.e. to detect and isolate some faults). Such sets of places are named "minimal sets of observable places" (Lefebvre 2004; Lefebvre et al., 2007).The problem that is solved is to give necessary and sufficient conditions in order to decide if the unbiased observation of the marking variation for a set of places P_O leads to immediate estimation of $X(\theta)$.

The subset of places $P_O \subset P$ is called a set of observable places (SOP) for θ, if $X(\theta)$ can be estimated exactly (i.e. with no error) and immediately (i.e. with no delay) from the unbiased measurement of ΔM_O between two consecutive observations. The subset of places $P_O \subset P$ is called a minimal set of observable places (MSOP) for θ, if P_O is a SOP for θ, and if there is no subset of places $P' \subset P_O$, $P' \neq P_O$ that is also a SOP for θ.

A SOP for θ provides enough information to detect and isolate a firing in θ before the occurrence of any other event and a MSOP is a minimal SOP for inclusion. According to basic assumption in section 2.b, P_O is a SOP for θ means that for any vector $X \in \{0, 1\}^q$ such that $||X|| \leq 1$, the unbiased measurement of $\Delta M_O = C(P_O).W.X \in (Z_3)^c$ leads to immediate and exact estimation of vector $X(\theta) = B(\theta).X \in \{0, 1\}^b$.

Characterisations of SOP can be obtained with an enumeration of the partitions for P_O or equivalently with the columns of the observable part W_O of incidence matrix (Lefebvre 2006, Lefebvre et al., 2007). For any marking variation ΔM_O let us define the disjoint partition of set P_O as $PA(\Delta M_O) = (P^+(\Delta M_O), P^-(\Delta M_O), P^0(\Delta M_O))$ with $P^+(\Delta M_O) = \{P_i\} \subset P_O$ such that $\Delta m_i > 0$, $P^-(\Delta M_O) = \{P_i\} \subset P_O$ such that $\Delta m_i < 0$ and $P^0(\Delta M_O) = \{P_i\} \subset P_O$ such that $\Delta m_i = 0$. Let us also consider the set of transitions $E(PA(\Delta M_O)) \subset T$:

$$E(PA(\Delta M_O)) = \left(\bigcap_{P_i \in P^+(\Delta M_O)} {}^\circ P_i \right) \cap \left(\bigcap_{P_i \in P^+(\Delta M_O)} P_i^\circ \right) \cap \left(\overline{\bigcup_{P_i \in P^0(\Delta M_O)} {}^\circ P_i \cup P_i^\circ} \right) \tag{7}$$

where °P_i stands for the set of P_i - upstream transitions and P_i° stands for the set of P_i - downstream transitions. The subset $P_O \subset P$ is a SOP for θ if and only if characterisation 1 or equivalently 2 is satisfied (Lefebvre et al., 2007):

1. For each subset $\theta_k \subset T$, k = 1,…,b, there exist a list of r_k disjoint partitions $PA_O(i) = (P^+_O(i), P^-_O(i), P^0_O(i))$ of P_O, i = 1,…, r_k, such that $P^+_O(i) \cup P^-_O(i) \neq \varnothing$ and :

$$\bigcup_{i=1,..,r_k} E(PA_O(i)) = \theta_k$$

2. For each subset $\theta_k \subset T$, k = 1,…,b, and for any couple of transitions $T_\alpha \in \theta_k$, $T_\beta \notin \theta_k$ we have $w_O(\alpha) \neq 0$ and $w_O(\alpha) \neq w_O(\beta)$.

The preceding result leads to an algorithm of complexity q^2 that generates the exhaustive list $F(P_O)$ of groups of transitions θ_k with minimal cardinality for which P_O is a SOP. The reduction of the obtained list thanks to linear algebra can be obtained as a post processing (Lefebvre 2004).

Algorithm a

1. Initialise list F to be empty
2. While T is not empty do
3. Initialise subset θ_k to be empty
4. Select $T_j \in T$
5. Remove T_j from set T
6. If $w_O(j) \neq 0$, then
7. Add T_j to subset θ_k
8. For any $T_i \in T$, do
9. If $w_O(j) = w_O(i)$, then transition T_j is added to set θ_k and T_i is removed from set T
10. End for (step 8)
11. Add subset θ_k to the list F
12. End if (step 6)
13. End while (step 2)

A recursive algorithm based on a combinatory exploration of the PN subsets of places generates also the list $G(\theta_k)$ of all MSOP for θ_k. From a computational point of view, this non polynomial algorithm must be used with some precautions. But the complexity depends on the number of potential observable places, and not on the size of the whole PN. Thus, it is suitable even for large scale systems as long as the considered set of potential observable places remains small. In comparison with algorithms that partially expand the reachability graph, the complexity of our results does not depend on the size of that graph.

Let us consider again PN2 with $\theta_1 = \{T_1\}$. Applying the preceding characterisation (condition 1 or 2), it is easy to state that P_{O1} is not a SOP for θ_1, whereas P_{O2} is a SOP and also a MSOP for θ_1. Moreover, this characterization leads to the exhaustive list of MSOP for θ_1 : $G(\theta_1) = \{\{P_2\}, \{P_1, P_3\}\}$. It leads also to the exhaustive list of transitions for which P_{O1} is a SOP : $F(P_{O1}) = \{\{T_5\}, \{T_6\}, \{T_7\}, \{T_1, T_2\}, \{T_3, T_4\}\}$ and to the exhaustive list of transitions for which P_{O2} is a SOP : $F(P_{O2}) = \{\{T_1\}, \{T_2\}, \{T_7\}, \{T_4, T_5\}\}$. As a consequence, $\{P_2\}$ and $\{P_1, P_3\}$ are the two possible MSOP for single – step diagnosis.

4.3 Diagnosis with CR and DP

Causality relationships (CR) and directed paths (DP) in PN models (Lefebvre et al., 2005) can also be used for multi-steps diagnosis purposes. In that case, diagnosis is improved by considering that some transitions may be observable. For that purpose, the set T_N is divided into a set T_O of observable transitions and a set T_U of unobservable ones.

Let N and N′ be two nodes (i.e. places or transitions) of PN model. A CR exists from N′ to N if and only if the behaviour of the node N′ could affect the variable attached to node N. The CR size (referred as CR – rank in the following) can be understood as the number of places in the shortest causality relationship from transition T_k to place P_i or transition T_j, and as the number of transitions in the shortest causality relationship from place P_k to place P_i or transition T_j. When no causality relationship exists, the CR - rank equals infinity. The CR – rank of PN nodes in range $I = [r_{min}, r_{max}] \cup \{\infty\}$ is characterised by the matrix CR(I) as given in (8) (Lefebvre et al., 2005):

$$CR(I) = \begin{pmatrix} CR_{PP}(I) & CR_{PT}(I) \\ CR_{TP}(I) & CR_{TT}(I) \end{pmatrix} \in I^{(n+q)x(n+q)} \tag{8}$$

with $CR_{PP}(I) = CR_{PP}(P_i, P_k, I) \in I^{n \times n}$, $CR_{PT}(I) = CR_{PT}(P_i, T_k, I) \in I^{n \times q}$, $CR_{TP}(I) = CR_{TP}(T_j, P_k, I) \in I^{q \times n}$, $CR_{TT}(I) = CR_{TT}(T_j, T_k, I) \in I^{q \times q}$.

Similarly, a DP exists from N′ to N if and only if a token is able to move from N′ to N. A DP between two nodes is also a CR but a CR is not necessary a DP. The DP - rank of PN nodes in range $I = [r_{min}, r_{max}] \cup \{\infty\}$ is characterised by a matrix $DP(I) \in I^{(n+q) \times (n+q)}$ similar to CR(I) (Lefebvre et al., 2005). From a computational point of view, the determination of the CR and DP matrices results from polynomial algorithms of complexity $(r_{max} - r_{min}).n.q$. The CR and DP ranks are defined according to the table 2.

M(A,r)	$A=W_{PR}+W_{PO}$	$A=W_{PR}$
$(A.(W_{PR})^T)^r$	$CR_{PP}(P_i, P_k, I)$	$DP_{PP}(P_i, P_k, I)$
$(A.(W_{PR})^T)^r.A$	$CR_{PT}(P_i, T_k, I)$	$DP_{PT}(P_i, T_k, I)$
$(W_{PR})^T.(A.(W_{PR})^T)^r$	$CR_{TP}(T_j, P_k, I)$	$DP_{TP}(T_j, P_k, I)$
$((W_{PR})^T.A)^r$	$CR_{TT}(T_j, T_k, I)$	$DP_{TT}(T_j, T_k, I)$

Table 2. CR and DP characterisation (Lefebvre et al., 2005)

In the next, the set I will be omitted as long as $I = [0, \min(n, q)] \cup \{\infty\}$ because CR and DP ranks cannot exceed the number of places or transitions.

In order to evaluate the potential of a set of observable nodes $P_O \cup T_O$ for diagnosis purpose, let us define the influence areas $I_{CR}(T_k)$ and $I_{DP}(T_k)$ of failure transition T_k, and dependence areas $D_{CR}(N)$ and $D_{DP}(N)$ of node N. The set $I_{CR}(T_k)$ of nodes that are CR - sensitive with respect to the transition T_k is called the CR - influence area of T_k. This area is a subnet of PN defined as $I_{CR}(T_k) = <P_{ICR}(T_k), T_{ICR}(T_k), Pre_{ICR}(T_k), Post_{ICR}(T_k)>$ where $P_{ICR}(T_k) \subset P$ is the set of places P_i such that $CR_{PT}(P_i, T_k) < \infty$. $T_{ICR}(T_k) \subset T$ is the set of transitions T_j such that $CR_{TT}(T_j, T_k) < \infty$, $Pre_{ICR}(T_k)$ and $Post_{ICR}(T_k)$ are the restrictions of the pre - incidence and post – incidence applications limited to the sets $P_{ICR}(T_k)$ and $T_{ICR}(T_k)$. The DP - influence area $I_{DP}(T_k)$ is defined in a similar way. The CR - dependence area $D_{CR}(N)$ of the node N is also a

subnet of PN defined as $D_{CR}(N) = <P_{DCR}(N), T_{DCR}(N), Pre_{DCR}(N), Post_{DCR}(N)>$ where $T_{DCR}(N)$ and $P_{DCR}(N)$ are the sets of transitions and places that are likely to influence the node N through a causality relationship. The DP - dependence area $D_{DP}(N)$ is defined in a similar way. The characterisation of the sets $P_{ICR}(T_k)$, $T_{ICR}(T_k)$, $P_{IDP}(T_k)$, $T_{IDP}(T_k)$, $T_{DCR}(P_i)$, $T_{DCR}(T_j)$ $T_{DDP}(P_i)$, and $T_{DDP}(T_j)$ is given in table 3 according to the position of finite entries in columns or rows of CR and DP matrices.

	CR		DP	
$P_{I..}(T_k)$	CR_{PT}	k^{th} column	DP_{PT}	k^{th} column
$T_{I..}(T_k)$	CR_{TT}	k^{th} column	DP_{TT}	k^{th} column
$T_{D..}(P_i)$	CR_{PT}	i^{th} row	DP_{PT}	i^{th} row
$T_{D..}(T_j)$	CR_{TT}	j^{th} row	DP_{TT}	j^{th} row

Table 3. Influence and dependence areas (Lefebvre et al., 2005)

The CR and DP investigation is helpful for delayed diagnosis of systems modelled by PN, in the sense that it provides in a systematic way the relationships between a fault transition and other nodes of PN.

1. Let $N \in P_O \cup T_O$. A necessary condition such that the observation of node N contributes to the diagnosis of F_k is $N \in I_{CR}(T_k)$ (Lefebvre et al., 2005).
2. Let $N \in P_O \cup T_O$. A sufficient condition to detect and isolate the firing of the fault transition T_k with the observation of node N is $N \in I_{DP}(T_k)$ and $T_{DDP(PN/Tk)}(N) = \varnothing$ if N is a place or $T_{DDP(PN/Tk)}(N) = \{N\}$ if N is a transition in PN/T_k (i.e. PN where the transition T_k has been removed) (Lefebvre et al., 2005).

If the preceding propositions cannot be applied, the nodes that have to be observed at first are the ones with the smaller dependence areas including fault transition T_k. This choice consists to select sensors in order to be sensitive with respect to the smaller set of events.

4.4 Application

PN can be used to model and monitor batch or chemical processes, like the system represented in figure 5a (Lefebvre et al., 2007). This system is composed of a tank R that can be filled and emptyed according to the flows Q_{source} provided by the source and Q_{demand} required by the distribution network. The system has three logical actuators: the input valves V_1 and V_2 and the output valve V_3 with two states {open = 1, closed = 0}. The continuous state variable h corresponds to the tank level and is defined according to $S.dh/dt = D - A.(2.g.h)^{1/2}$ with S the tank section, A the output pipe section and g the gravity acceleration.

The goal of the PN supervisor PN4 is to keep the level h below the treshold LSH+ and above the treshold LSH- in order to limit the pressure in distribution network. When LSH- is reached V_1 is opened during an appropriate time to fill the tank. Then V_1 is closed. Eventually V_2 is closed and V_3 is opened if LSHH is reached. Two logical level sensors are used to detect the tresholds LSH- and LSHH.

$$CR_{PT}(PN4)=\begin{array}{c} \begin{matrix} T_1 & T_2 & T_3 & T_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}, \qquad DP_{PT}(PN4/T_1)=\begin{array}{c} \begin{matrix} T_2 & T_3 & T_4 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ \infty & \infty & \infty \\ \infty & 0 & \infty \end{pmatrix} \end{array} \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix} \tag{9}$$

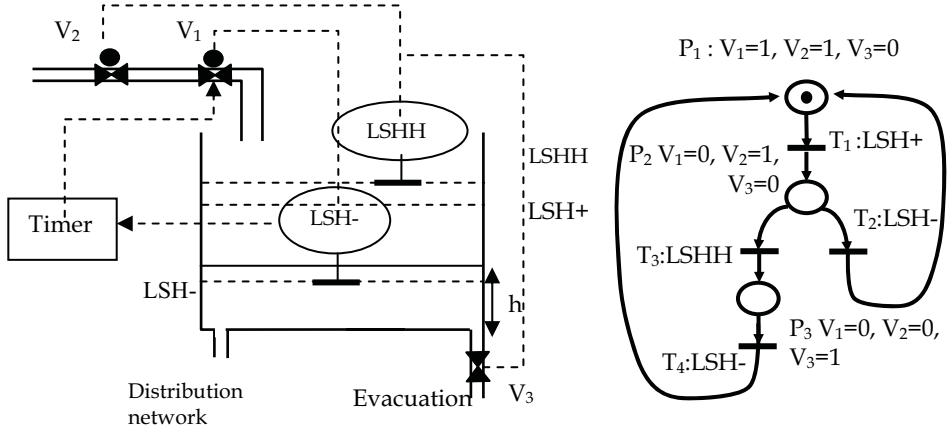


Fig. 5. Tank system a) Sens a) l actuators; b) PN4 model of the controller b)

The set of observable places is assumed to be defined as $P_O = \{P_3\}$. A single fault is considered when the treshold LSH+ is exceeded. The MSOP for LSH+, included in P, are given by $G(\{LSH+\}) = \{\{P_1\}, \{P_2\}\}$. The resulting MSOP are not suitable because no sensor is used to detect the threshold LSH+. Matrix CR_{PT} shows that $P_{ICR}(T_1) = \{P_1, P_2, P_3\}$: the observation of each place contributes to the diagnosability of PN4. Matrix DP_{PT} in $PN4/T_1$, shows that $T_{DDP(PN4/T1)}(P_2) = \varnothing$ (but P_2 is not observable) and $T_{DDP(PN4/T1)}(P_3) = \{T_3\}$. As a conclusion, the observation of P_3 can be used as a two-steps diagnoser to detect a fault of sensor LSH+.

5. Diagnosis based on algebraic approaches

5.1 Diagnosis based on coding theory

Event sequences estimation is an important issue for fault diagnosis of DES, so far as fault events cannot be directly measured. This section is about event sequences estimation with PN models. Events are assumed to be represented with transitions and firing sequences are estimated from measurements of the marking variation. Estimation with and without measurement errors can be discussed in n – dimensional vector space over alphabet Z_3 (Lefebvre, 2006; Lefebvre, 2007). The basic idea to correct measurement errors by projecting measurements in orthogonal subspace of Vect(W) where Vect(W) stands for the subspace generated by the columns of W. This method is inspired from linear coding theory (Van Lint, 1999) and extend the results presented for continuous PN in (Lefebvre et al., 2001).

Measurement $\Delta\hat{M}$ of marking variation $\Delta M \in (Z_3)^n$ may be affected by an additive error vector $E \in (Z_3)^n$: $\Delta\hat{M} = \Delta M + E$. The error vector will be characterized according to the Hamming distance d(W) of the considered PN that is defined with the Hamming distance of the columns of incidence matrix :

$$d(W) = \min\{\min\{d(w_i, w_j), i \neq j\}, \min\{d_0(w_i)\}\} \quad (10)$$

where $d(w_i, w_j)$ stands for the Hamming distance between columns w_i and w_j of matrix W and $d_0(w_i) = d(w_i, 0)$ stands for the weight of vector w_i.

It is assumed that error vector E verifies the following conditions:

1. $Pr(d_0(E) = 0) > Pr(d_0(E) = 1) > ... > Pr(d_0(E) = n)$ where $Pr(d_0(E) = i)$ is the probability that weight of E equals i;
2. An error in position i does not influence other positions;
3. A symbol in error can be each of the remaining symbols with equal probability.

A short estimation algorithm easy to use and to implement when state measurement is complete (i.e. all entries of $\Delta\hat{M}$ are measured), and error free (i.e. measurement equals actual marking variation ΔM), is based on the comparison of the measurement with respect to columns of W and zero vector (this corresponds to the condition of event-detectability in case that all places are observable). When this measurement equals a single column of W, the algorithm decides that the corresponding transition fired. When it equals the zero vector, the algorithm decides that no transition fired.

When measurement is perturbed by non zero error E, two problems must be mentioned:

1. A miss estimation may occur when $\Delta\hat{M}$ is non zero and different from any columns of W. The estimation algorithm is not able to decide if a transition fired or not and which transition fired. As consequence the algorithm does not give any decision.
2. A wrong estimation may occur when $\Delta\hat{M}$ does not equal actual marking variation ΔM but equals zero vector or another column of W. The estimation algorithm decides if a transition fired or not and which transition fired, but the decision is wrong due to the measurement error.

To overcome these difficulties and to improve estimation, diagnosis can be reformulated as a linear problem in $((Z_3)^n, +, *)$, with the Smith transformation of W, where "+" and "*" stand for the sum and product endowed over Z_3. The Smith transformation results from elementary operations (i.e. row or column permutations, linear combinations and external products), summed up in matrices $P \in (Z_3)^{n \times n}$ and $Q \in (Z_3)^{q \times q}$ such that:

$$P * W * Q = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix} \quad (11)$$

I_r is the identity matrix of dimension r x r, and r is the rank of matrix W. The Smith transformation leads to reduced incidence matrix W' defined as in equation (12) :

$$W' = (I_r\ 0) * Q^{-1} = (I_r\ 0) * P * W = F * W \in (\mathbf{Z_3})^{r \times q} \quad (12)$$

Necessary and sufficient conditions for firing sequences estimation can be stated when measurement is error free and basic assumption in section 2.2 is satisfied: columns of incidence matrix W' defined by equation (12) are distinct and non zero (Lefebvre, 2006).
In case of measurement errors that satisfy assumptions 1 to 3, two sets of sufficient conditions for firing sequences estimation have been also proposed (Lefebvre, 2007):

1. Columns of incidence matrix W are distinct, non zero and errors E that disturb satisfy $d_0(E) \leq (d(W) - 1) / 2$ (i.e. the number of disturbed entries of measurement is no larger than $(d(W) - 1) / 2$).
2. Columns of incidence matrix W' defined by equation (12) are distinct and non zero, and considered errors E belongs to distinct cosets different from C(0). The coset C(u) of u is defined as $C(u) = \{x \in (Z_3)^n$ such that $x = u + y$ with $y \in Vect(W)\}$, for any vector $u \in (Z_3)^n$.

Moreover, the use of the Smith transformation of incidence matrix is also helpful to define the parity check matrix $H^T = (0\ I_{n-r})^{(n-r) \times n} * P \in (Z_3)^{(n-r) \times n}$, and to work out the syndrome of marking variation measurements $S(\Delta\hat{M}) = H^T * \Delta\hat{M}$ and to compare it with the syndrome of marking variation errors $S(E) = H^T * E$. As a consequence the method leads to a less complex and more efficient diagnosis algorithm for DES modeled with PN (algorithm c) in comparison with usual method based on Hamming distance (algorithm b).

Algorithm b

1. For each time k, measure $\hat{M}(k)$ the current state of DES
2. Compute $\Delta\hat{M}(k) = \hat{M}(k) - \hat{M}(k-1)$
3. Compute weight $d_0(\Delta\hat{M}(k))$. If $d_0(\Delta\hat{M}(k)) \leq (d(W) - 1) / 2$, then no event occurs between two consecutive state measurements. Go to step 6.
4. Compute Hamming distance $d(\Delta\hat{M}(k), w_j)$ for each column w_j of W. If $d(\Delta\hat{M}(k), w_j) \leq (d(W) - 1) / 2$ then T_j fired. Go to step 6.
5. If for all $j = 1,...,q$, $d(\Delta\hat{M}(k), w_j) > (d(W) - 1) / 2$ then measurement is too much disturbed by errors (i.e. $d_0(E) > (d(W) - 1) / 2$) and no decision is provided (i.e. a miss estimation occurs).
6. Wait until time k + 1. Go to step 1.

Algorithm c

1. For each time k, measure $\hat{M}(k)$ the current state of DES
2. Compute $\Delta\hat{M}(k) = \hat{M}(k) - \hat{M}(k-1)$
3. Compute $H^T * \Delta\hat{M}(k)$. If $H^T * \Delta\hat{M}(k) = 0$ then measurement is not disturbed by errors: $\Delta M(k) = \Delta\hat{M}(k)$. Go to step 5.
4. If syndrome $H^T * \Delta\hat{M}(k) \neq 0$, compute coset leader E(k) and $\Delta M(k) = \Delta\hat{M}(k) - E(k)$. Go to step 5.
5. Compute $\Delta M'(k) = F * \Delta M(k)$.
6. Compare $\Delta M'(k)$ with zero vector. If $\Delta M'(k) = 0$ then no event occurs between 2 consecutive state measurements. Go to step 8.
7. Compare $\Delta M'(k)$ with columns of matrix W'. If $\Delta M'(k) = w'_j$ then T_j fired. Go to step 8.
8. Wait until time k + 1. Go to step 1.

The correction capacity (i.e. number of error vectors that are corrected) of algorithm b is given by equation (13):

$$\sum_{i=1}^{(d(W)-1)/2} 2^i.\left(\frac{n!}{i!(n-i)!}\right) \tag{13}$$

and its complexity results from 2n.(q+1) scalar comparisons or operations whereas correction capacity of algorithm c equals $3^{n-r} - 1$, and its complexity results from $r.(2n+q)+(n-r).(2n-1+3^{n-r})$ scalar comparisons or operations (Lefebvre, 2007).
As a conclusion, one can notice that algorithm c is more efficient for PN with small rank in comparison with the number of places, and that it is of particular interest for PN with few transitions in comparison with the number of places. Another conclusion is to prefer algorithm c for PN with a small Hamming distance. This result is not surprising as long as the correction capacity of algorithm a is directly related to the value of Hamming distance. For PN with small Hamming distance, the number of biased markings that belong to a single sphere is also small.

5.2 Redundant Petri nets embedding

This method incorporates redundancy into Petri nets and uses algebraic decoding techniques as the Berlekamp – Massey decoding (Berlekamp, 1984) to detect and identify faults (Li et al., 2004; Wu et al., 2005). The marking of the original PN is embedded into a redundant one and the diagnosis of faults is performed by mean of linear parity checks. The algorithm operates in the integer finite field of order p, referred as (Z^+_p) with p a prime integer large enough. This approach has a complexity of $m^2.(n+q)$ (Wu et al., 2002) improved to complexity m.(n+q) (Wu et al., 2005) where 2.m represent the number of places that are added to the original PN.
In comparison with the method in section 5.1, two kinds of faults are considered : (1) place faults are associated with conditions that cause the corruption of the number of tokens in some places of the PN. Place faults are measured, with the Hamming distance metric, in terms of the number of faulty places independent of the number of erroneous tokens in each faulty place ; (2) transition failures are associated with preconditions that prevent tokens from being removed from the input places in some transitions (even though tokens are deposited at the corresponding output places) or postconditions that prevent tokens from being deposited at the output places in some transitions (even though tokens are removed from the corresponding input places). Errors "-1" and "+1" are used respectively and transitions faults are measured with the Lee distance metric (Berlekamp, 1984). By adding 2.m places in the redundant PN, Wu et al. proves that the method allows the simultaneous identification of m place faults and 2.m – 1 transition failures.
It is assumed that the firing of the transitions in the redundant PN are not directly observable whereas the marking is periodically observed, and that the diagnosis is performed over a time interval of N sampling periods (N is eventually chosen equal to 1). It is also assumed that

1. A particular transition does not suffer both a precondition and postcondition during interval [1, N] (otherwise, their effect will be cancelled);
2. The erroneous number of tokens in each place is also bounded within interval [-(p - 1/2), (p - 1/2)];
3. Parameter p is a prime integer that satisfy p > max(n + 2.m, q).

The key idea of the method is to design two matrices $C \in (Z^+_p)^{2m \times n}$ and $D \in (Z^+)^{2m \times n}$ that define the state of the embedded PN such that equation (14) is satisfied :

$$M(k+1) = M(k) + \begin{pmatrix} W_{PO} \\ C.W_{PO} - D \end{pmatrix}.X(k) - \begin{pmatrix} W_{PR} \\ C.W_{PR} - D \end{pmatrix}.X(k) \tag{14}$$

Defining the parity check matrix as in $P = (-C\ I_{2m})$, the syndrome of marking M(k) is given by $S(k) = P.M(k)$.
Let us define $E_T^+(N) \in (Z^+)^q$ as the vector of postcondition faults, $E_T^-(N) \in (Z^+)^q$ as the vector of precondition faults and $E_P(N) \in (Z^+)^{2m+n}$ as the place faults vector during time interval [1, N]. As a consequence the faulty marking is defined as :

$$\hat{M}(N) = M(N) - \begin{pmatrix} W_{PO} \\ C.W_{PO} - D \end{pmatrix}.E_T^+(N) + \begin{pmatrix} W_{PR} \\ C.W_{PR} - D \end{pmatrix}.E_T^-(N) + E_P(N) \tag{15}$$

The identification of both transition failures and place faults based on the syndrome *S(N)* is completely determined by matrices *D* and *C* :

$$S(N) = D.(E_T^+(N) - E_T^-(N)) + P.E_P(N) \tag{16}$$

On one hand, Wu et al. propose to define matrix *D* as in equation (17) :

$$D_{2m} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_q \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_q^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2m-1} & \alpha_2^{2m-1} & \alpha_3^{2m-1} & \cdots & \alpha_q^{2m-1} \end{pmatrix} \in \left(Z^+\right)^{2m \times q} \tag{17}$$

where α_i are q distinct non zero elements in Z^+_p. In case m = 1, the determination of matrix D is given according to :

$$D_2 = \begin{pmatrix} 1 & 2 & 3 & \cdots & q \\ 1 & 2^2 & 3^2 & \cdots & q^2 \end{pmatrix} \bmod p \in \left(Z^+\right)^{2 \times q}$$

On the other hand, they propose to define matrix C such that equation (18) is satisfied (operations are defined in Z^+_p) :

$$\Phi .(-C\ I_{2m}) = H_{2m} \tag{18}$$

with :

$$H_{2m} = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_{n+2m} \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_{n+2m}^2 \\ \alpha_1^3 & \alpha_2^3 & \alpha_3^3 & \cdots & \alpha_{n+2m}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2m} & \alpha_2^{2m} & \alpha_3^{2m} & \cdots & \alpha_{n+2m}^{2m} \end{pmatrix} \in \left(Z^+\right)^{2m \times (n+2m)}$$

(19)

$$\Phi = \begin{pmatrix} \alpha_{n+1} & \alpha_{n+2} & \alpha_{n+3} & \cdots & \alpha_{n+2m} \\ \alpha_{n+1}^2 & \alpha_{n+2}^2 & \alpha_{n+3}^2 & \cdots & \alpha_{n+2m}^2 \\ \alpha_{n+1}^3 & \alpha_{n+2}^3 & \alpha_{n+3}^3 & \cdots & \alpha_{n+2m}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n+1}^{2m} & \alpha_{n+2}^{2m} & \alpha_{n+3}^{2m} & \cdots & \alpha_{n+2m}^{2m} \end{pmatrix} \in \left(Z^+\right)^{2m \times 2m}$$

In order to identify simultaneously place and fault transition Wu et al. define :

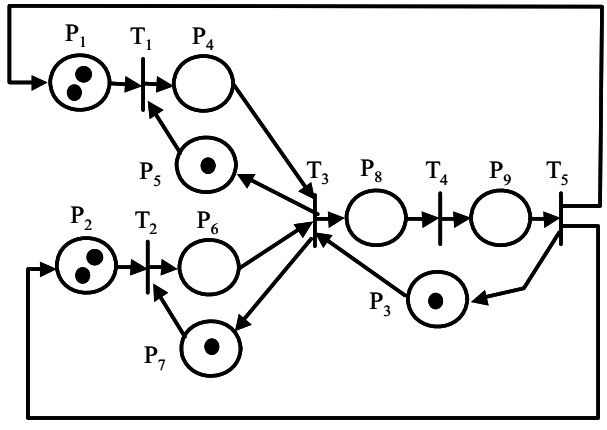
$$\begin{aligned} D^* &= -p.D \\ C^* &= p.1 - C \\ P^* &= (C - p.1\, I_{2m}) \end{aligned} \tag{20}$$

where 1 is a 2m x n matrix with all entries being 1. The syndrome S(N) defined as S(N) = P^*.M(N) is used to identify first m or less place faults by means of the Berlekamp – Massey algorithm and then 2m – 1 transitions by computing the modified syndrome :

$$S_T(N) = (S(N) - P^*.E_p(N)) \,/\, p = D.(E_T^+(N) - E_T^-(N)) \tag{21}$$

5.3 Applications

Algebraic methods have been used for the diagnosis of manufacturing and robotic systems (Lefebvre, 2007, Wu et al., 2005) and for large scale power networks like the IEEE 118-bus power system (Ren et al., 2006).



$$W = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

Fig. 6. PN5 model of a manufacturing system

In order to illustrate algebraic methods, let us consider PN5 in figure 6, that is a simplified model of a manufacturing workshop (Silva et al., 2004). The final product is composed of two different parts that are processed in two separate machines modelled by transitions T_1 and T_2, and stored in buffers P_4 and P_6, respectively. Then, they are

assembled by the machine T_3, and processed in T_4 and T_5. During the processing, several tools are needed, modelled by places P_3, P_5 and P_7.
PN5 has n = 9 places, q = 5 transitions, is of rank r = 4 and incidence matrix W has a Hamming distance d = 2. Matrices F and H^T, worked out as in section 5.1, are given according to equation (22):

$$F = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad H^T = \begin{pmatrix} -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{22}$$

Syndromes	Errors of weight 1	Syndromes	Errors of weight 1
$(-1\ 0\ 0\ 1\ 0)^T$	$(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(1\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0)^T$
$(1\ 0\ 0\ -1\ 0)^T$	$(-1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(-1\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 0)^T$
$(0\ 1\ -1\ 0\ 0)^T$	$(0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ 1\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0)^T$
$(0\ -1\ 1\ 0\ 0)^T$	$(0\ -1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ -1\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 0)^T$
$(1\ -1\ 1\ -1\ 1)^T$	$(0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ 1\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0)^T$
$(-1\ 1\ -1\ 1\ -1)^T$	$(0\ 0\ -1\ 0\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ -1\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 0\ 0\ -1\ 0\ 0)^T$
$(0\ 0\ 0\ 1\ 0)^T$	$(0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ 1)^T$	$(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0)^T$ $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)^T$
$(0\ 0\ 0\ -1\ 0)^T$	$(0\ 0\ 0\ -1\ 0\ 0\ 0\ 0\ 0)^T$	$(0\ 0\ 0\ 0\ -1)^T$	$(0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0)^T$ $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1)^T$

Table 4. Correspondence between syndromes and coset leaders for PN5

PN5 has 243 cosets and each coset has 81 vectors. The table 4 gives the relationships between syndromes and coset leaders. Let us notice that the two last syndromes correspond to two different coset leaders. As a consequence not all errors of weight 1 will be corrected by algorithms b and c (errors $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0)^T$ and $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)^T$ cannot be separated as errors $(0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0)^T$ and $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1)^T$).
Analysis of performance and numerous simulations show that the miss estimation rate for algorithm c is quite better in comparison with algorithm b (Lefebvre 2007), but the wrong estimation rate for c increases in comparison with b. Let us mention that whatever the algorithm used, the presence of miss estimation depends strongly on the Hamming distance of W.
Applying the method developed in 5.2 in order to identify 1 place fault and 1 transition failure (m = 1) with p = 13, the matrices D and D*, that lead to transition failure diagnosis are given according to equation (23):

$$D = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 9 & 3 & 12 \end{pmatrix}, \quad D^* = -13 \text{ x } D = \begin{pmatrix} -13 & -26 & -39 & -52 & -65 \\ -13 & -52 & -117 & -39 & -156 \end{pmatrix} \tag{23}$$

On the other hand, the matrices H, C and C*, that lead to place fault diagnosis are given according to equations (24) and (25):

$$H = \begin{pmatrix} 5 & 10 \\ 12 & 9 \end{pmatrix} \begin{pmatrix} 4 & 10 & 2 & 10 & 5 & 2 & 11 & 11 & 4 & 1 & 0 \\ 2 & 3 & 2 & 1 & 3 & 10 & 10 & 6 & 8 & 0 & 1 \end{pmatrix} \tag{24}$$

$$C = \begin{pmatrix} 4 & 10 & 2 & 10 & 5 & 2 & 11 & 11 & 4 \\ 2 & 3 & 2 & 1 & 3 & 10 & 10 & 6 & 8 \end{pmatrix}$$

$$C^* = \begin{pmatrix} 9 & 3 & 11 & 3 & 8 & 11 & 2 & 2 & 9 \\ 11 & 10 & 11 & 12 & 10 & 3 & 3 & 7 & 5 \end{pmatrix} \tag{25}$$

The parity check matrix is defined according to (26) :

$$P^* = \begin{pmatrix} -9 & -3 & -11 & -3 & -8 & -11 & -2 & -2 & \_9 & 1 & 0 \\ -11 & -10 & -11 & -12 & -10 & -3 & -3 & -7 & -5 & 0 & 1 \end{pmatrix} \tag{26}$$

As a consequence 2 places are added in the PN model of figure 6 for diagnosis purposes. These new places are defined according to equation (27):

$$C^*.W_{PO} - D^* = \begin{pmatrix} 16 & 37 & 51 & 61 & 88 \\ 25 & 55 & 137 & 44 & 188 \end{pmatrix}, \quad C^*.W_{PR} - D^* = \begin{pmatrix} 30 & 31 & 64 & 54 & 74 \\ 34 & 65 & 143 & 46 & 161 \end{pmatrix} \tag{27}$$

and the initial marking of embedded PN is given according to (28):

$$M_I^* = \begin{pmatrix} I_9 \\ C^* \end{pmatrix} M_I = \begin{pmatrix} 2 & 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 45 & 66 \end{pmatrix}^T \tag{28}$$

The use of embedded PN defined with equations (27) and (28) is useful to detect at first place faults and then transitions failures according to the comparison of syndrome S(N) = P*.M(N) with the columns of matrices H given by equation (24) and with the columns of matrix D given by (23).

6. Conclusions

The investigation of diagnosis methods for discrete event systems shows that Petri nets is efficient not only to model the considered systems but also to support the diagnosis methods. Several approaches can be used in order to check diagnosability, to select sensors and to work out diagnosers. The table 5 sums up the main characteristics of these method.

As a conclusion it is important to notice the great effort, observed this last years to develop and improve diagnosis methods for DES. The strong connection with observation properties in automata and the use of advances in computer science like the coding theory have played an important role in that development. Now, the challenges are, from our

point of view, to continue this investigation, by combining the different methods together and also to take advantages from many important contributions that have been proposed for continuous systems. To build a bridge from continuous variable systems to DES theories remains one of the most promising issues for the next years.

	State based approach (section 3)	Event detectability and SOP (sections 4.1 and 4.2)	CR and DP investigation (section 4.3)	Diagnosis in Z_3 (section 5.1)	Diagnosis in Z^+_p (section 5.2)
Diagnosability checking	Yes	Yes	Yes	No	No
Sensor selection	No	Yes	Yes	No	No
Immediate / delayed diagnosis	Immediate and delayed diagnosis at most in k steps	Immediate diagnosis	Delayed diagnosis at least in k steps	Immediate diagnosis	Immediate and delayed diagnosis at most in N steps
Marking measurements	Partial and unbiased	Partial and unbiased	X	Partial and biased	Complete and biased
Partial firing sequence observation	Yes	No	X	No	No
Complexity : Polynomial or Non polynomial	NP	P / NP	P	P	P

Table 5. Main features of several diagnosis methods for DES

7. References

Alcaraz-Mejia M., Lopez-Mellado E., Ramirez-Trevino A., Rivera-Rangel I. (2003). Petri net based fault diagnosis of DES, *Proc. IEEE-SMC03*, pp. 4730-4735, Washington, USA.

Aramburo-Lizarraga, J.; Lopez-Mellado, E.; Ramirez-Trevino, A. (2005). Distributed fault diagnosis using Petri net reduced models**,** *Proc. IEEE-SMC05*, vol. 1, pp. 702-705.

Askin R.G., Standridge C. R. (1993). *Modeling and analysis of Petri nets*, John Wiley and sons Inc.

Berlekamp R.E. (1984). *Algebraic coding theory*, Laguna Hills, CA, Aegean Park.

Benveniste A., Fabre F., Jard C., Haar S. (2003). Diagnosis of asynchronous discrete event systems, a net unfolding approach, *Trans. IEEE –TAC*, vol. 48, no.5.

Blanke M. (1996). Consistent design of dependable control systems, *Control Engineering Practice*, vol. 4, no. 9, pp. 1305 – 1312.

Blanke M., Kinnaert M., Lunze J., Staroswiecki M. (2003). *Diagnosis and fault tolerant control*, Springer Verlag, New York.

Cassandras C.G., Lafortune S. (1999). *Introduction to discrete event systems,* Kluwer Academic Pub.

Chung S.L, Wu C.C., Jeng M. (2003). Failure diagnosis: a case study on modeling and analysis by Petri nets, *Proc. IEEE-SMC03*, pp. 2727-2732, Washington, USA.

Cordier M.O., Dague P., Lévy F., Dumas M., Montmain J. Staroswiecky M., Travé-Massuyès L. (2000). AI and automatic control approaches of model-based diagnosis : links and underlying hypothesis, *Proc. IFAC Symposium on fault detection, Supervision and Safety for Technical Processes*, pp. 274 – 279, Budapest, Hungary.

David R., Alla H. (1992). *Petri nets and grafcet – tools for modelling discrete events systems*, Prentice Hall, London.

Gertler J. (1998). *Fault detection and diagnosis in engineering systems*, Marcel Dekker, New York.

Giua A., Seatzy C. (2002). Observability of place / transition nets, *Trans. IEEE – TAC*, vol. 47, no. 9, pp. 1424 – 1437.

Ichikawa, A., Hiraishi, K. (1988). Analysis and Control of Discrete Event Systems Represented by Petri Nets, *Proc. IIASA Conf.*, pages 115-134. Springer-Verlag, Berlin, West Germany.

Lefebvre D., El Moudni A. (2001). Firing and enabling sequences estimation for timed Petri nets, *Trans. IEEE - SMCA*, vol. 31, no.3, pp. 153- 162.

Lefebvre D., Delherm C. (2005). Diagnosis with causality relationships and directed paths in Petri net models, *Proc. IFAC WC05*, Prague, Czech Republic.

Lefebvre D. (2004). About estimation problems with Petri net models for fault detection and isolation with discrete event and hybrid systems, *Proc. SAUM04*, Invited lecture, pp. 42 – 51, Beograd, Serbia and Montenegro.

Lefebvre D., Delherm C., Leclercq E., Druaux F. (2006). Some contributions with Petri nets for the modelling, analysis and control of HDS, *Proc. ICHSA*, Invited lecture, Lafayette, Louisiana, USA.

Lefebvre D. (2006a). Sensoring and diagnosis of DES with Petri net models, *Proc. IFAC Safeprocess 2006*, invited session "Model based fault analysis during a system's entire life cycle", pp. 1213 - 1218, Beijing, China.

Lefebvre D. (2006b) Firing sequences estimation for ordinary Petri nets, *Proc. Workshop IAR - ACD*, Nancy, France.

Lefebvre D., Delherm C. (2007). Fault detection and isolation of discrete event systems with Petri net models, *Trans. IEEE – TASE*, vol. 4, no. 1, pp. 114 – 118.

Lefebvre D. (2007). Firing sequences estimation in vector space over Z_3 for ordinary Petri nets, *Trans. IEEE – SMCA*, under review.

Li L., Hadjicostis C. N. Sreenivas R. S. (2004). Fault Detection and Identification in Petri Net Controllers, *Proc. IEEE-CDC04*, pp. 5248 – 5253, Atlantis, Paradise Island, Bahamas.

Patton R.J., Frank M., Clark R.N. (Eds), (1989). *Fault diagnosis in dynamic systems theory and applications*, Prentice Hall, New York.

Patton R.J., Frank M., Clark R.N. (Eds), (1999). Iss*ues of fault diagnosis for dynamical systems*, Springer Verlag, London.

Ramirez-Trevino A., Ruiz-Bletran E., Rivera-Rangel I., Lopez-Mellado E. (2004). Diagnosability of discrete event systems. A Petri net based approach, *Proc. IEEE-ICRA*, pp. 541 – 546.

Ramirez-Trevino A., Ruiz-Bletran E., Rivera-Rangel I., Lopez-Mellado E. (2007). Online Fault Diagnosis of Discrete Event Systems. A Petri Net-Based Approach, *Trans. IEEE – TASE*, vol. 4, no. 1, pp. 31-39.

Rausand M., Hoyland A. (2004). *System reliability theory : models, statistical methods, and applications*, Wiley, Hoboken, New Jersey.

Ren H., Mi Z. (2006). Power system fault diagnosis modeling techniques based on encoded Petri nets, *Proc. IEEE Power Engineering Society General Meeting*.

Sampath M., Sengupta R., Lafortune S., Sinnamohideen K., Teneketzis D. (1995). Diagnosibility of discrete event systems, *Trans. IEEE-TAC*, vol. 40, no.9, pp. 1555-1575.

Silva M., Recalde L. (2004). On fluidification of Petri Nets: from discrete to hybrid and continuous models, *Annual Reviews in Control,* vol. 28, no. 2, pp. 253-266.

Ushio T., Onishi I., Okuda K., (1998). Fault detection based on Petri net models with faulty behaviours, *Proc. IEEE – SMC98*, pp 113-118.

Van Lint J.H. (1999). *Introduction to Coding Theory*, Graduate Texts in Mathematics, vol. 86, Springer Verlag.

Wen Y.L, Li C.H., Jeng M. (2005). A polynomial algorithm for checking diagnosability of Petri nets, *Proc. IEEE-SMC05*, pp. 2542-2547, vol. 3.

Wu Y., Hadjicostis N. (2002). Non-concurrent fault identification in discrete event systems using encoded Petri net states, *Proc. IEEE – CDC02*, vol. 4, pp4018-4023.

Wu Y., Hadjicostis N. (2005). Algebraic approaches for fault identification in discrete event systems, *Trans. IEEE - TAC*, vol. 50, no. 12, pp. 2048 – 2053.

17

Augmented Marked Graphs and the Analysis of Shared Resource Systems

King Sing Cheung
University of Hong Kong
Hong Kong

1. Introduction

Augmented marked graphs were first introduced in 1997 (Chu & Xie, 1997). They are not well known as compared to other sub-classes of Petri nets such as free-choice nets (Desel & Esparza, 1995), and the properties of augmented marked graphs are not studied extensively. However, augmented marked graphs possess a structure which is desirable for modelling shared resources, and for this reason, they are often used in modelling shared resource systems, such as manufacturing systems (Chu & Xie, 1997; Zhou & Venkatesh, 1999; Jeng et al., 2000; Jeng et al., 2002; Huang et al., 2003; Cheung & Chow, 2005).

There are a few published works on augmented marked graphs. Based on siphons and mathematical programming, Chu and Xie proposed a necessary and sufficient condition of live and reversible augmented marked graphs, which checks the existence of potential deadlocks (Chu & Xie, 1997). However, this involves the flow of tokens during execution and cannot be checked simply by looking into the structure. Chu and Xie also proposed another siphon-based characterisation for live and reversible augmented marked graphs but it provides a sufficient condition only. On the other hand, the boundedness and conservativeness of augmented marked graphs were not investigated.

In the literature, apart from (Chu & Xie, 1997), the studies of augmented marked graphs mainly focus on their property-preserving synthesis or composition. Jeng et al. proposed a synthesis of process nets (covering augmented marked graphs) for manufacturing system design, where the condition of liveness and reversibility are based on siphons and the firability of transitions (Jeng et al., 2000; Jeng et al., 2002). Huang et al. also investigated the composition of augmented marked graphs so that properties such as liveness, boundedness and reversibility can be preserved (Huang et al., 2003).

In our earlier works on augmented marked graphs, we proposed characterisations for their liveness, boundedness, reversibility and conservativeness and applied the characterisations to the analysis and design of manufacturing systems, object-oriented systems and shared resource systems (Cheung, 2004; Cheung, 2005; Cheung & Chow, 2005a; Cheung & Chow, 2005b; Cheung & Chow, 2005c; Cheung, 2006; Cheung & Chow, 2006; Cheung et al., 2006; Cheung, 2007). This paper consolidates our previous works with a special focus on the properties of augmented marked graphs.

First, we provide a number of characterisations for live and reversible augmented marked graphs, based on siphons and cycles. In particular, a property called R-inclusion property is

introduced to characterise the siphon-trap property of augmented marked graphs. With this property, the liveness and reversibility of an augmented marked graph can be analysed through cycles instead of siphons. Second, we introduce R-transformation which transforms an augmented marked graph into a marked graph. Its boundedness and conservativeness can be determined by checking the cycles of the transformed marked graph. Based on these characterisations, some pretty simple conditions and procedures are derived for checking the liveness, reversibility, boundedness and conservativeness of an augmented marked graph. These will be illustrated using the dining philosophers problem. We then show the application to manufacturing system design.

The rest of this paper is organised as follows. Following this introduction, Section 2 provides the preliminaries to be used in this paper. Section 3 describes augmented marked graphs and their known properties. Section 4 shows characterisations for liveness and reversibility while Section 5 shows characterisations for boundedness and conservativeness. Section 6 then illustrates these characterisations using the dining philosophers example. Section 7 describes the application to manufacturing system design. Finally, Section 8 concludes this paper with a brief discussion.

2. Preliminaries

This section provides the preliminaries to be used in this paper for those readers who are not familiar with Petri nets (Peterson, 1981; Reisig, 1985; Murata, 1989; Desel & Reisig, 1998).

Definition 2.1. A place-transition net (PT-net) is a 4-tuple $N = \langle P, T, F, W \rangle$, where P is a set of places, T is a set of transitions, $F \subseteq ((P \times T) \cup (T \times P))$ is a flow relation, and $W : F \rightarrow \{ 1, 2, ... \}$ is a weight function. N is said to be ordinary if and only if the range of W is $\{ 1 \}$.

An ordinary PT-net is usually written as $\langle P, T, F \rangle$. In the rest of this paper, unless specified otherwise, all PT-nets refer to ordinary PT-nets.

Definition 2.2. Let $N = \langle P, T, F, W \rangle$ be a PT-net. For any $x \in (P \cup T)$, ${}^\bullet x = \{ y \mid (y, x) \in F \}$ and $x^\bullet = \{ y \mid (x, y) \in F \}$ are called the pre-set and post-set of x, respectively.

For clarity in presentation, the pre-set and post-set of a set of places or transitions $X = \{ x_1, x_2, ..., x_n \}$ can be written as ${}^\bullet X$ and $X^\bullet$ respectively, where ${}^\bullet X = {}^\bullet x_1 \cup {}^\bullet x_2 \cup ... \cup {}^\bullet x_n$ and $X^\bullet = x_1^\bullet \cup x_2^\bullet \cup ... \cup x_n^\bullet$.

Definition 2.3. For a PT-net $N = \langle P, T, F, W \rangle$, a path is a sequence of places and transitions $\rho = \langle x_1, x_2, ..., x_n \rangle$, such that $(x_i, x_{i+1}) \in F$ for i = 1, 2, ..., n-1. ρ is said to be elementary if and only if it contains no duplicate places or transitions.

Definition 2.4. For a PT-net $N = \langle P, T, F, W \rangle$, a sequence of places $\langle p_1, p_2, ..., p_n \rangle$ is called a cycle if and only if there exists a set of transitions $\{ t_1, t_2, ..., t_n \}$, such that $\langle p_1, t_1, p_2, t_2, ..., p_n, t_n \rangle$ forms an elementary path and $(t_n, p_1) \in F$.

Definition 2.5. For a PT-net $N = \langle P, T, F, W \rangle$, a marking is a function $M : P \rightarrow \{ 0, 1, 2, ...\}$, where M(p) is the number of tokens in p. (N, M_0) represents N with an initial marking M_0.

Definition 2.6. For a PT-net (N, M_0), a transition t is said to be enabled at a marking M if and only if $\forall\ p \in {}^\bullet t : M(p) \geq W(p,t)$. On firing t, M is changed to M' such that $\forall\ p \in P : M'(p) = M(p) - W(p,t) + W(t,p)$. In notation, $M\ [N,t\rangle\ M'$ or $M\ [t\rangle\ M'$.

Definition 2.7. For a PT-net (N, M_0), a sequence of transitions $\sigma = \langle t_1, t_2, ..., t_n \rangle$ is called a firing sequence if and only if $M_0\ [t_1\rangle ... [t_n\rangle\ M_n$. In notation, $M_0\ [N,\sigma\rangle\ M_n$ or $M_0\ [\sigma\rangle\ M_n$.

Definition 2.8. For a PT-net (N, M_0), a marking M is said to be reachable if and only if there exists a firing sequence σ such that $M_0 [\sigma\rangle M$. In notation, $M_0 [N,*\rangle M$ or $M_0 [*\rangle M$. $[N, M_0\rangle$ or $[M_0\rangle$ represents the set of all reachable markings of (N, M_0).
Definition 2.9. Let $N = \langle P, T, F, W \rangle$ be a PT-net, where $P = \{ p_1, p_2, ..., p_m \}$ and $T = \{ t_1, t_2, ..., t_n \}$. The incidence matrix of N is an $m \times n$ matrix V whose typical entry $v_{ij} = W(p_i,t_j) - W(t_j,p_i)$ represents the change in number of tokens in p_i after firing t_j once, for i = 1, 2, ..., m and j = 1, 2, ..., n.
Definition 2.10. For a PT-net (N, M_0), a transition t is said to be live if and only if $\forall M \in [M_0\rangle, \exists M' : M [*\rangle M' [t\rangle$. (N, M_0) is said to be live if and only if every transition is live.
Definition 2.11. For a PT-net (N, M_0), a place p is said to be k-bounded if and only if $\forall M \in [M_0\rangle : M(p) \leq k$, where k is a positive integer. (N, M_0) is said to be bounded if and only if every place is k-bounded, and safe if and only if every place is 1-bounded.
Definition 2.12. A PT-net (N, M_0) is said to be reversible if and only if $\forall M \in [M_0\rangle : M [*\rangle M_0$.
Definition 2.13. A PT-net $N = \langle P, T, F, W \rangle$ is said to be conservative if and only if there exists a $|P|$-vector $\alpha > 0$ such that $\alpha V = 0$, where V is the incidence matrix of N.
Property 2.1. A PT-net is bounded if it is conservative (Murata, 1989; Desel & Reisig, 1998).
Definition 2.14. For a PT-net $N = \langle P, T, F, W \rangle$, a place invariant is a $|P|$-vector $\alpha \geq 0$ such that $\alpha V = 0$, where V is the incidence matrix of N.
Definition 2.15. For a PT-net N, a set of places S is called a siphon if and only if ${}^\bullet S \subseteq S^\bullet$. S is said to be minimal if and only if there does not exist another siphon S' in N such that $S' \subset S$.
Definition 2.16. For a PT-net, a set of places T is called a trap if and only if $T^\bullet \subseteq {}^\bullet T$.
Definition 2.17. A PT-net (N, M_0) is said to satisfy the siphon-trap property if and only if every siphon contains a marked trap (or every minimal siphon contains a marked trap).
Definition 2.18. A marked graph is an ordinary PT-net $N = \langle P, T, F, W \rangle$ such that $\forall p \in P : |{}^\bullet p| = |p^\bullet| = 1$.
Property 2.2. A marked graph (N, M_0) is live if and only if every cycle is marked by M_0 (Reisig, 1985; Murata, 1989).
Property 2.3. A live marked graph (N, M_0) is bounded if and only if every place belongs to a cycle marked by M_0 (Reisig, 1985; Murata, 1989).
Property 2.4. A live and bounded marked graph is reversible (Reisig, 1985; Murata, 1989).
Property 2.5. For a marked graph, the corresponding place vector of a cycle is a place invariant (Reisig, 1985; Murata, 1989).

3. Augmented marked graphs

This section describes augmented marked graphs and their known properties on liveness and reversibility.
Definition 3.1. An augmented marked graph $(N, M_0; R)$ is a PT-net (N, M_0) with a specific subset of places R, satisfying the following conditions : (a) Every place in R is marked by M_0. (b) The net (N', M_0') obtained from $(N, M_0; R)$ by removing the places in R and their associated arcs is a marked graph. (c) For each $r \in R$, there exist $k_r \geq 1$ pair of transitions $D_r = \{ \langle t_{s1}, t_{h1}\rangle, \langle t_{s2}, t_{h2}\rangle, ..., \langle t_{skr}, t_{hkr}\rangle \}$, such that $r^\bullet = \{ t_{s1}, t_{s2}, ..., t_{skr} \} \subseteq T$ and ${}^\bullet r = \{ t_{h1}, t_{h2}, ..., t_{hkr} \} \subseteq T$ and that, for each $\langle t_{si}, t_{hi}\rangle \in D_r$, there exists in N' an elementary path ρ_{ri} connecting t_{si} to t_{hi}. (d) In (N', M_0'), every cycle is marked and no ρ_{ri} is marked.

Figure 1 shows an augmented marked graph (N, M_0; R), where R = { r_1, r_2 }. For r_1, D_{r1} = { ⟨t_1, t_{11}⟩, ⟨t_3, t_9⟩ }. For r_2, D_{r2} = { ⟨t_2, t_{11}⟩, ⟨t_4, t_{10}⟩ }.

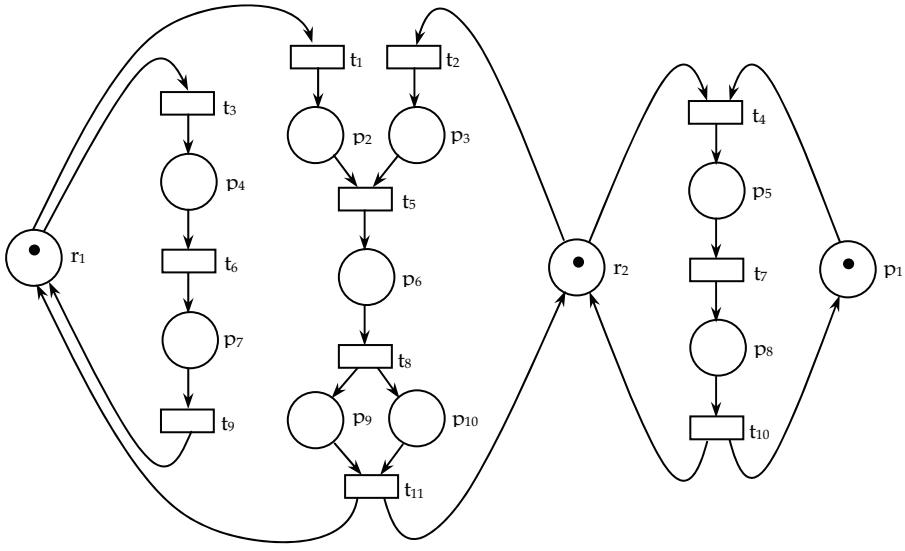


Fig. 1. A typical augmented marked graph.

Property 3.1. An augmented marked graph is live if and only if it does not contain any potential deadlock - a siphon which would eventually become empty (Chu & Xie, 1997).
Property 3.2. An augmented marked graph is reversible if it is live (Chu & Xie, 1997).
Property 3.3. An augmented marked graph is live and reversible if and only if every minimal siphon would never become empty.
Proof. (⇐) It follows from Properties 3.1 and 3.2 that an augmented marked graph is live and reversible if every minimal siphon would never become empty. (⇒) It follows from Property 3.1 that every minimal siphon would never become empty.
Property 3.4. An augmented marked graph (N, M_0; R) is live and reversible if every minimal siphon, which contains at least one place of R, contains a marked trap (Chu & Xie, 1997).
For the augmented marked graph (N, M_0; R) in Figure 1, the minimal siphons include : { p_1, p_5, p_8}, { r_1, p_2, p_4, p_6, p_7, p_9 }, { r_1, p_2, p_4, p_6, p_7, p_{10} }, { r_2, p_3, p_5, p_6, p_8, p_9 } and { r_2, p_3, p_5, p_6, p_8, p_{10} }. Each of these minimal siphons contains a marked trap, and would never become empty. (N, M_0; R) is live and reversible.

4. Liveness and reversibility

This section provides characterisations for live and reversible augmented marked graphs, based on siphons and cycles. Strategies for checking liveness and reversibility are then presented.
Definition 4.1. For a PT-net N, Ω_N is defined as the set of all cycles in N.
Definition 4.2. Let N be a PT-net. For a subset of cycles $Y \subseteq \Omega_N$, P[Y] is defined as the set of places in Y, and $T[Y] = {}^\bullet P[Y] \cap P[Y]^\bullet$ is defined as the set of transitions generated by Y.

For clarity in presentation, P[{γ}] and T[{γ}] can be written as P[γ] and T[γ], to denote the set of places in a cycle γ and the set of transitions generated by γ, respectively.

Definition 4.3. For a PT-net N, an elementary path $\rho = \langle x_1, x_2, ..., x_n \rangle$ is said to be conflict-free if and only if, for any transition x_i in ρ, $j \neq (i-1) \Rightarrow x_j \notin {}^\bullet x_i$ (Barkaoui, 1995).

Lemma 4.1. Let S be a minimal siphon of a PT-net. For any p, p' ∈ S, there exists in S a conflict-free path from p to p' (Barkaoui, 1995).

Property 4.1. For a minimal siphon S of an augmented marked graph (N, M_0; R), there exists a set of cycles $Y \subseteq \Omega_N$ such that P[Y] = S.

Proof. Let $S = \{ p_1, p_2, ..., p_n \}$. For each p_i, it follows from the definition of augmented marked graphs that ${}^\bullet p_i \neq \emptyset$. Then, there exists $p_j \in S$, where $p_j \neq p_i$, such that $(p_j^\bullet \cap {}^\bullet p_i) \neq \emptyset$. According to Lemma 4.1, p_i connects to p_j via a conflict-free path in S. Since p_j connects to p_i, this forms a cycle γ_i in S, where $p_i \in P[\gamma_i] \subseteq S$. Let $Y = \{ \gamma_1, \gamma_2, ..., \gamma_n \}$. We have $P[Y] = P[\gamma_1] \cup P[\gamma_2] \cup ... \cup P[\gamma_n] \subseteq S$. On the other hand, $S \subseteq (P[\gamma_1] \cup P[\gamma_2] \cup ... \cup P[\gamma_n]) = P[Y]$ since $S = \{ p_1, p_2, ..., p_n \}$. Hence, P[Y] = S.

Property 4.2. Every cycle in an augmented marked graph is marked.

Proof. (by contradiction) Let (N, M_0; R) be an augmented marked graph. Suppose there exists a cycle γ in (N, M_0; R), such that γ is not marked. Then, γ does not contain any place in R. Hence, γ also exists in the net (N', M_0') obtained from (N, M_0; R) after removing the places in R and their associated arcs. However, by definition of augmented marked graphs, every cycle in (N', M_0') is marked.

Property 4.3. Every siphon in an augmented marked graph is marked.

Proof. For an augmented marked graph, according to Properties 4.1 and 4.2, every minimal siphon contains cycles and is marked. Hence, every siphon, which contains at least one minimal siphon, is marked.

Property 4.4. Let (N, M_0; R) be an augmented marked graph. For every r ∈ R, there exists a minimal siphon which contains only one marked place r.

Proof. Let $D_r = \{ \langle t_{s1}, t_{h1} \rangle, \langle t_{s2}, t_{h2} \rangle, ..., \langle t_{sn}, t_{hn} \rangle \}$, where $r^\bullet = \{ t_{s1}, t_{s2}, ..., t_{sn} \}$ and ${}^\bullet r = \{ t_{h1}, t_{h2}, ..., t_{hn} \}$. By definition of augmented marked graphs, for each $\langle t_{si}, t_{hi} \rangle \in D_r$, t_{si} connects to t_{hi} via an elementary path ρ_i which is not marked. Let $S = P_1 \cup P_2 \cup ... \cup P_n \cup \{ r \}$, where P_i is the set of places in ρ_i. We have ${}^\bullet P_i \subseteq (P_i^\bullet \cup r^\bullet)$ because, for each $p \in P_i$, $| {}^\bullet p | = | p^\bullet | = 1$. Then, $({}^\bullet P_1 \cup {}^\bullet P_2 \cup ... \cup {}^\bullet P_n) \subseteq (P_1^\bullet \cup P_2^\bullet \cup ... \cup P_n^\bullet \cup r^\bullet)$. Besides, ${}^\bullet r = \{ t_{h1}, t_{h2}, ..., t_{hn} \} \subseteq (P_1^\bullet \cup P_2^\bullet \cup ... \cup P_n^\bullet)$. Hence, ${}^\bullet S = ({}^\bullet P_1 \cup {}^\bullet P_2 \cup ... \cup {}^\bullet P_n \cup {}^\bullet r) \subseteq (P_1^\bullet \cup P_2^\bullet \cup ... \cup P_n^\bullet \cup r^\bullet) = S^\bullet$. Therefore, S is a siphon, in which r is the only one marked place. Let S' be a minimal siphon in S. According to Property 4.3, S' is marked. Since r is the only one marked place in S, r is also the only one marked place in S'.

Consider the augmented marked graph (N, M_0; R) shown in Figure 1. Every minimal siphon is covered by cycles. For example, for a minimal siphon $S_1 = \{ r_1, p_2, p_4, p_6, p_7, p_9 \}$, there exists a set of cycles $Y_1 = \{ \gamma_{11}, \gamma_{12} \}$, where $\gamma_{11} = \langle r_1, p_4, p_7 \rangle$ and $\gamma_{12} = \langle r_1, p_2, p_6, p_9 \rangle$, such that $S_1 = P[Y_1]$. For another minimal siphon $S_2 = \{ r_2, p_3, p_5, p_6, p_8, p_{10} \}$, there exists a set of cycles $Y_2 = \{ \gamma_{21}, \gamma_{22} \}$, where $\gamma_{21} = \langle r_2, p_5, p_8 \rangle$ and $\gamma_{22} = \langle r_2, p_3, p_6, p_{10} \rangle$, such that $S_2 = P[Y_2]$. S_1 is a minimal siphon, in which $r_1 \in R$ is the only one marked place. Also, S_2 is a minimal siphon, in which $r_2 \in R$ is the only one marked place.

Definition 4.4. For an augmented marked graph (N, M_0; R), a minimal siphon is called a R-siphon if and only if it contains at least one place in R.

Definition 4.5. For an augmented marked graph (N, M_0; R), a minimal siphon is called a NR-siphon if and only if it does not contain any place in R.

Definition 4.6. Let N be a PT-net. For a set of places Q in N, $\Omega_N[Q]$ is defined as the set of cycles that contains at least one place in Q.

For clarity in presentation, $\Omega_N[\{p\}]$ can be written as $\Omega_N[p]$ to denote the set of cycles that contains a place p.

Figure 2 shows an augmented marked graph (N, M_0; R), where R = { r_1, r_2 }. There are five minimal siphons, namely, S_1 = { r_1, p_3, p_4, p_7, p_8 }, S_2 = { r_1, p_3, p_5, p_7, p_8 }, S_3 = { r_2, p_2, p_4, p_6, p_8, p_9, p_{10} }, S_4 = { r_2, p_2, p_5, p_6, p_8, p_9, p_{10} } and S_5 = { p_1, p_3, p_7 }. S_1, S_2, S_3 and S_4 are R-siphons while S_5 is a NR-siphon.

Property 4.5. For an augmented marked graph (N, M_0; R), a R-siphon is covered by a set of cycles $Y \subseteq \Omega_N[R]$.

Proof. (By contradiction) Let S be a R-siphon. By Property 4.1, S is covered by cycles. Suppose there exists a cycle γ in S, such that $\gamma \notin \Omega_N[R]$. According to the definition of augmented marked graphs, for any $p \in P[\gamma]$, $| {}^\bullet p | = | p^\bullet | = 1$. Hence, ${}^\bullet P[\gamma] = P[\gamma]^\bullet$, and $P[\gamma]$ is also a siphon. Since there exists a place $r \in R$ such that $r \in S$ but $r \notin P[\gamma]$, we have $P[\gamma] \subset S$. However, by definition of minimal siphons, there does not exists any siphon $S' = P[\gamma]$ in S, such that $S' = P[\gamma] \subset S$.

For the augmented marked graph (N, M_0; R) shown in Figure 2. every R-siphon is covered by a set of cycles in $\Omega_N[R]$. For example, a R-siphon S_1 = { r_1, p_3, p_4, p_7, p_8 } is covered by a set of cycles Y_1 = { γ_{11}, γ_{12} } $\subseteq \Omega_N[R]$, where γ_{11} = ⟨ r_1, p_3, p_7 ⟩ and γ_{12} = ⟨ r_1, p_4, p_8 ⟩. Another R-siphon S_2 = { r_1, p_3, p_5, p_7, p_8 } is covered by a set of cycles Y_2 = { γ_{21}, γ_{22} } $\subseteq \Omega_N[R]$, where γ_{21} = ⟨ r_1, p_3, p_7 ⟩ and γ_{22} = ⟨ r_1, p_5, p_8 ⟩.

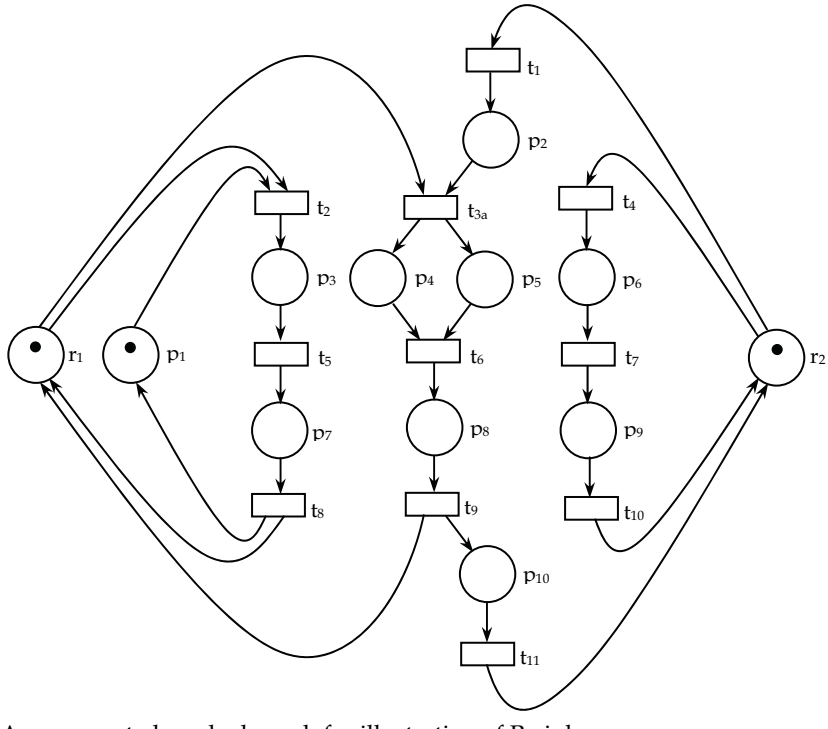


Fig. 2. An augmented marked graph for illustration of R-siphons.

Property 4.6. Let S be a R-siphon of an augmented marked graph $(N, M_0; R)$. For every $t \in (S^\bullet \setminus {}^\bullet S)$, there does not exist any $s \in (S \setminus R)$ such that $t \in s^\bullet$.

Proof. (by contradiction) Suppose s exists. By definition of augmented marked graphs, $| {}^\bullet s | = | s^\bullet | = 1$. S is covered by cycles in accordance with Property 4.5. Hence, t is the one and only one transition in $s^\bullet$, where $t \in T[Y] = (S^\bullet \cap {}^\bullet S)$. This however contradicts $t \in (S^\bullet \setminus {}^\bullet S)$. □

Property 4.7. For an augmented marked graph $(N, M_0; R)$, a NR-siphon contains itself as a marked trap and would never become empty.

Proof. Let S be a NR-siphon. According to Property 4.3, S is marked. It follows from the definition of augmented marked graphs that, for any $s \in S$, $| {}^\bullet s | = | s^\bullet | = 1$. Then, ${}^\bullet S = S^\bullet$ and S is also a trap. Hence, S contains itself as a marked trap and would never become empty.

Property 4.8. An augmented marked graph $(N, M_0; R)$ is live and reversible if and only if no R-siphons eventually become empty.

Proof. ($\Leftarrow$) According to Property 4.7, a NR-siphon would never become empty. Given that no R-siphons eventually become empty, every minimal siphon would never become empty. According to Property 3.3, $(N, M_0; R)$ is live and reversible. ($\Rightarrow$) It follows from Property 3.1 that no R-siphons eventually become empty.

Property 4.9. An augmented marked graph $(N, M_0; R)$ satisfies the siphon-trap property if and only if every R-siphon contains a marked trap.

Proof. ($\Leftarrow$) According to Property 4.7, a NR-siphon contains a marked trap. Given that every R-siphon contains a marked trap, every minimal siphon contains a marked trap. ($\Rightarrow$) Every minimal siphon contains marked trap implies that every R-siphon contains a marked trap. □

Consider the augmented marked graph $(N, M_0; R)$ shown in Figure 2. Every R-siphon contains marked trap. Each of the R-siphons $S_1 = \{ r_1, p_3, p_4, p_7, p_8 \}$, $S_2 = \{ r_1, p_3, p_5, p_7, p_8 \}$, $S_3 = \{ r_2, p_2, p_4, p_6, p_8, p_9, p_{10} \}$ and $S_4 = \{ r_2, p_2, p_5, p_6, p_8, p_9, p_{10} \}$ contains a marked trap and would never become empty. Also, the NR-siphon $S_5 = \{ p_1, p_3, p_7 \}$ contains itself as a marked trap and would never become empty. $(N, M_0; R)$ is live and reversible.

Property 4.10. (characterisation of Property 3.4) An augmented marked graph $(N, M_0; R)$ is live and reversible if every R-siphon contains a marked trap.

Proof. For $(N, M_0; R)$, if every R-siphon contains a marked trap, according to Property 4.9, the siphon-trap property is satisfied. Hence, every minimal siphon would never become empty. It then follows from Property 3.3 that $(N, M_0; R)$ is live and reversible.

Property 4.8 provides a simpler necessary and sufficient condition for live and reversible augmented marked graphs, as compared to Properties 3.1 and 3.2. According to Property 4.8, only R-siphons are considered. Typically, for an augmented marked graph $(N, M_0; R)$, the set R is small, so only a small number of siphons need to be considered. Properties 3.4 and 4.10 also refers to the same set of siphons but only a sufficient condition is provided.

With Properties 4.8 and 4.10, we can determine if an augmented marked graph is live and reversible based on R-siphons. On the other hand, Property 4.5 provides a characterisation for R-siphons so that R-siphons can be easily identified by finding cycles in $\Omega_N[R]$.

We may now derive the following strategy for checking the liveness and reversibility of an augmented marked graph $(N, M_0; R)$:

1. Find all R-siphons based on $\Omega_N[R]$.
2. Check if every R-siphon contains a marked trap. If yes, report $(N, M_0; R)$ is live and reversible. Otherwise, go to (c).

3. For each R-siphon which does not contain any marked trap, check if it would never become empty. If yes, (N, M_0; R) is live and reversible. Otherwise, (N, M_0; R) is neither live nor reversible.

In the following, a property called R-inclusion is introduced for characterising the liveness and reversibility of augmented marked graphs.

Definition 4.7. For a PT-net N, a set of cycles $Y \subseteq \Omega_N$ is said to be conflict-free if and only if, for any q, q' ∈ P[Y], there exists in P[Y] a conflict-free path from q to q'.

For the PT-net N shown in Figure 3, consider three cycles $\gamma_1, \gamma_2, \gamma_3 \in \Omega_N[p_3]$, where $\gamma_1 = \langle p_3, p_2, p_7 \rangle$, $\gamma_2 = \langle p_3, p_4 \rangle$ and $\gamma_3 = \langle p_3, p_1, p_6, p_{10}, p_8 \rangle$. $Y_1 = \{ \gamma_1, \gamma_2 \}$ is conflict-free because for any q, q' ∈ P[Y_1], there exists in P[Y_1] a conflict-free path from q to q'. $Y_2 = \{ \gamma_2, \gamma_3 \}$ is not conflict-free. Consider Y_2. We have $p_4, p_8 \in P[Y_2]$. p_4 is connected to p_8 via only one path $\rho = \langle p_4, t_5, p_3, t_1, p_1, t_3, p_6, t_6, p_{10}, t_9, p_8 \rangle$ in Y_2, and ρ is not conflict-free because $p_4, p_8 \in {}^\bullet t_5$.

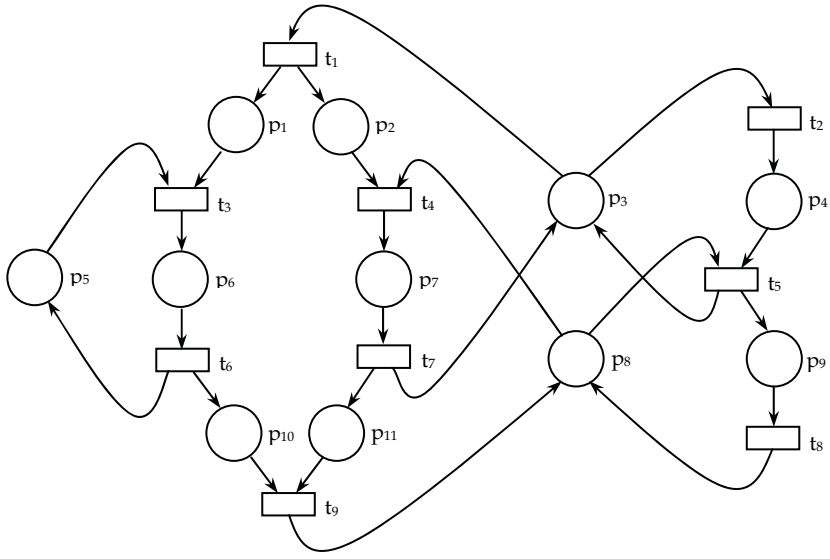


Fig. 3. A PT-net N for illustration of conflict-free cycles.

Property 4.11. Let S be a minimal siphon of an augmented marked graph (N, M_0; R), and $Y \subseteq \Omega_N$ be a set of cycles such that S = P[Y]. Then, Y is conflict-free.

Proof. It follows from Lemma 4.1 that, for any q, q' ∈ S = P[Y], there exists in S = P[Y] a conflict-free path from q to q'. Hence, Y is conflict free.

For the augmented marked graph shown in Figure 1, { $r_1, p_2, p_4, p_6, p_7, p_9$ } is a minimal siphon covered by a set of cycles { $\langle r_1, p_4, p_7 \rangle, \langle r_1, p_2, p_6, p_9 \rangle$ } which is conflict free. { $r_2, p_3, p_5, p_6, p_8, p_{10}$ } is another minimal siphon covered by a set of cycles { $\langle r_2, p_5, p_8 \rangle, \langle r_2, p_3, p_6, p_{10} \rangle$ } which is conflict-free. For the augmented marked graph shown in Figure 2, { r_1, p_3, p_4, p_7, p_8 } is a minimal siphon covered by a set of cycles { $\langle r_1, p_3, p_7 \rangle, \langle r_1, p_4, p_8 \rangle$ } which is conflict free. { r_1, p_3, p_5, p_7, p_8 } is another minimal siphon covered by a set of cycles { $\langle r_1, p_3, p_7 \rangle, \langle r_1, p_5, p_8 \rangle$ } which is conflict free.

Definition 4.8. Let (N, M_0; R) be an augmented marked graph. A place $r \in R$ is said to satisfy the R-inclusion if and only if, for any set of cycles $Y \subseteq \Omega_N[R]$ such that Y is conflict-free, ${}^\bullet r \subseteq T[Y] \Rightarrow r^\bullet \subseteq T[Y]$.

Figure 4 shows an augmented marked graph (N, M_0; R), where R = { r_1, r_2 }. Consider r_1. For any set of cycles $Y_1 \subseteq \Omega_N[R]$ such that Y_1 is conflict-free, ${}^\bullet r_1 \subseteq T[Y_1] \Rightarrow r_1^\bullet \subseteq T[Y_1]$. Next, consider r_2. For any set of cycles $Y_2 \subseteq \Omega_N[R]$ such that Y_2 is conflict-free, ${}^\bullet r_2 \subseteq T[Y_2] \Rightarrow r_2^\bullet \subseteq T[Y]$. Both r_1 and r_2 satisfy the R-inclusion.

Figure 5 shows an augmented marked graph (N, M_0; R), where R = { r_1, r_2 }. For r_1, there exists a set of cycles $Y_1 = \{ \gamma_{11}, \gamma_{12} \} \subseteq \Omega_N[R]$, where $\gamma_{11} = \langle r_1, p_5 \rangle$ and $\gamma_{12} = \langle r_1, p_5, r_2, p_6 \rangle$. ${}^\bullet r_1 = \{ t_5, t_6 \} \subseteq T[Y_1] = \{ t_3, t_4, t_5, t_6 \}$, but $r_1^\bullet = \{ t_2, t_3 \} \not\subseteq T[Y_1]$. For r_2, there exists a set of cycles $Y_2 = \{ \gamma_{21}, \gamma_{22} \} \subseteq \Omega_N[R]$, where $\gamma_{21} = \langle r_2, p_6 \rangle$ and $\gamma_{22} = \langle r_2, p_6, r_1, p_5 \rangle$. ${}^\bullet r_2 = \{ t_5, t_6 \} \subseteq T[Y_2] = \{ t_3, t_4, t_5, t_6 \}$, but $r_2^\bullet = \{ t_1, t_4 \} \not\subseteq T[Y_2]$. Hence, r_1 and r_2 do not satisfy the R-inclusion property.

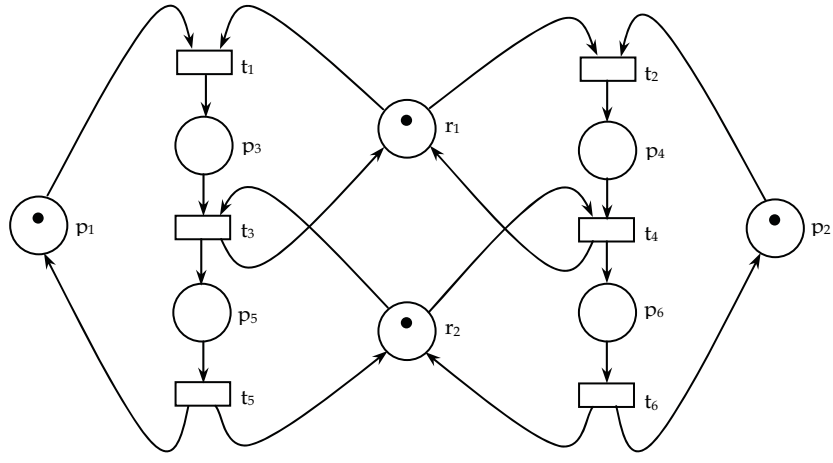


Fig. 4. An augmented marked graph for illustration of R-inclusion.

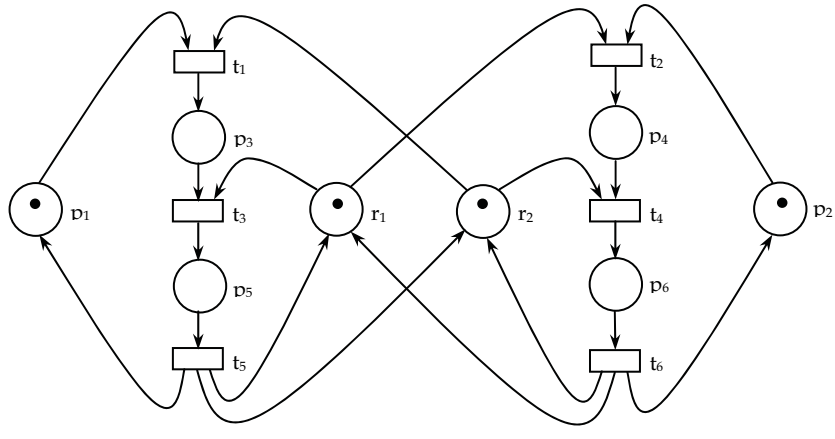


Fig. 5. Another augmented marked graph for illustration of R-inclusion.

Property 4.12. For an augmented marked graph $(N, M_0; R)$, a R-siphon S contains itself as a marked trap if every place $r \in R$ in S satisfies the R-inclusion property.

Proof. Let $S = \{ p_1, p_2, ..., p_n \}$. According to Property 4.3, S is marked. It follows from Properties 4.5 and 4.11 that there exists a set of cycles $Y \subseteq \Omega_N[R]$, such that Y is conflict-free and $P[Y] = S$. Since S is a siphon, for each $p_i \in S$, ${}^\bullet p_i \subseteq ({}^\bullet S \cap S^\bullet) = ({}^\bullet P[Y] \cap P[Y]^\bullet) = T[Y]$. In case $p_i \notin R$, $p_i^\bullet \subseteq T[Y]$ because $| {}^\bullet p_i | = | p_i^\bullet | = 1$. In case $p_i \in R$, given that p_i satisfies the R-inclusion property, $p_i^\bullet \subseteq T[Y]$. Every $p_i^\bullet \subseteq T[Y] = ({}^\bullet P[Y] \cap P[Y]^\bullet)$ and $p_i^\bullet \subseteq {}^\bullet P[Y] = {}^\bullet S$. Since $S^\bullet = (p_1^\bullet \cup p_2^\bullet \cup ... \cup p_n^\bullet) \subseteq {}^\bullet S$, S is also a trap. S contains itself as a marked trap. □

Consider the augmented marked graph $(N, M_0; R)$, where $R = \{ r_1, r_2 \}$, shown in Figure 4. Both r_1 and r_2 satisfy the R-inclusion property. $\{ r_1, p_3, p_4 \}$ is a minimal siphon which contains itself as a marked trap. $\{ r_2, p_5, p_6 \}$ is another minimal siphon which contains itself as a marked trap.

Property 4.13. An augmented marked graph $(N, M_0; R)$ satisfies the siphon-trap property if and only if every place $r \in R$ satisfies the R-inclusion property.

Proof. ($\Leftarrow$) It follows from Properties 4.12 and 4.9. ($\Rightarrow$ by contradiction) Suppose there exists $r \in R$, such that r does not satisfy the R-inclusion property. According to Property 4.4, there exists a R-siphon S, in which r is the only marked place. It follows from Properties 4.5 and 4.11 that there exists $Y \subseteq \Omega_N[R]$, such that Y is conflict-free and $S = P[Y]$. According to Property 4.9, S contains a marked trap Q. Then, $r \in Q$ and $r^\bullet \subseteq ({}^\bullet Q \cap Q^\bullet)$. Since S is a siphon, we have ${}^\bullet r \subseteq ({}^\bullet S \cap S^\bullet) = ({}^\bullet P[Y] \cap P[Y]^\bullet) = T[Y]$. However, as r does not satisfy the R-inclusion property, $r^\bullet \not\subseteq T[Y] = ({}^\bullet P[Y] \cap P[Y]^\bullet) = ({}^\bullet S \cap S^\bullet)$, implying $r^\bullet \not\subseteq ({}^\bullet Q \cap Q^\bullet)$. □

Property 4.14. An augmented marked graph $(N, M_0; R)$ is live and reversible if every place $r \in R$ satisfies the R-inclusion property.

Proof. According to Property 4.13, $(N, M_0; R)$ satisfies the siphon-trap property. It follows from Property 4.10 that $(N, M_0; R)$ is live and reversible.

Consider the augmented marked graph $(N, M_0; R)$, where $R = \{ r_1, r_2 \}$, shown in Figure 4. Both r_1 and r_2 satisfy the R-inclusion property. $(N, M_0; R)$ satisfies the siphon-trap property, and is live and reversible.

Property 4.14 provides a cycle-based sufficient condition for live and reversible augmented marked graphs. Without finding siphons and checking if each of these siphons contains a marked trap, we need to check the R-inclusion property which involves finding cycles and checking their pre-sets and post-sets. This provides an alternative characterisation for live and reversible augmented marked graphs, apart from the existing siphon-based ones.

Based on Properties 4.5, 4.8, 4.10, 4.12 and 4.14, we may revise the strategy for checking the liveness and reversibility of an augmented marked graph $(N, M_0; R)$ with the use of the R-inclusion property, as follows.

1. Check if every $r \in R$ satisfies the R-inclusion property. If yes, report $(N, M_0; R)$ is live and reversible. Otherwise go to (b).
2. Let $R' \subseteq R$ be the set of places which do not satisfy the R-inclusion property. Based on $\Omega_N[R']$, find all R-siphons which contain at least one place in R'.
3. For each R-siphon identified in (b), check if it contains a marked trap. If yes, $(N, M_0; R)$ is live and reversible. Otherwise, go to (d).
4. For each R-siphon identified in (b) that does not contain any marked trap, check if it would never become empty. If yes, $(N, M_0; R)$ is live and reversible. Otherwise, $(N, M_0; R)$ is neither live nor reversible.HHHHHH

5. Boundedness and conservativeness

This section first introduces a transformation, called R-transform, for augmented marked graphs. Based on R-transform, a number of characterisations for bounded and conservative augmented marked graphs are then derived. Strategies for checking boundedness and conservativeness are then presented.

Property 5.1. Let (N, M_0; R) be an augmented marked graph to be transformed into (N', M_0') as follows. For each place r ∈ R, where D_r = { ⟨t_{s1}, t_{h1}⟩, ⟨t_{s2}, t_{h2}⟩, ..., ⟨t_{skr}, t_{hkr}⟩ }, replace r with a set of places { p_1, p_2, ..., p_{kr} }, such that $M_0'[p_i] = M_0[r]$ and $p_i^\bullet$ = { t_{si} } and $^\bullet p_i$ = { t_{hi} } for i = 1, 2, ..., k_r. Then, (N', M_0') is a marked graph.

Proof. According to the definition of augmented marked graphs, for each place p ∉ R in N, M_0; R), | $^\bullet$p | = | p$^\bullet$ | = 1. Each place r ∈ R is replaced by a set of places { p_1, p_2, ..., p_{kr} }, where | $^\bullet p_i$ | = | $p_i^\bullet$ | = 1 for i = 1, 2, ..., k_r. Hence, for every place q in N', | $^\bullet$q | = | q$^\bullet$ | = 1. (N', M_0') is a marked graph.

Definition 5.1. Let (N, M_0; R) be an augmented marked graph. The marked graph (N', M_0') obtained from (N, M_0; R) after the transformation as stated in Property 5.1 is called the R-transform of (N, M_0; R).

Property 5.2. The R-transform of an augmented marked graph is live.

Proof. Let (N', M_0') be the R-transform of an augmented marked graph (N, M_0; R). Since the transformation process does not create new cycles, cycles in (N', M_0') also exist in (N, M_0; R). According to Property 4.1, every cycle in (N, M_0; R) is marked, and hence, every cycle in (N', M_0') is marked. Since (N', M_0') is a marked graph, it follows from Property 2.2 that (N', M_0') is live.

Figure 6 shows an augmented marked graph (N, M_0; R). Figure 7 shows the R-transform of (N, M_0; R), where r is replaced by { q_1, q_2 }. It is a live marked graph.

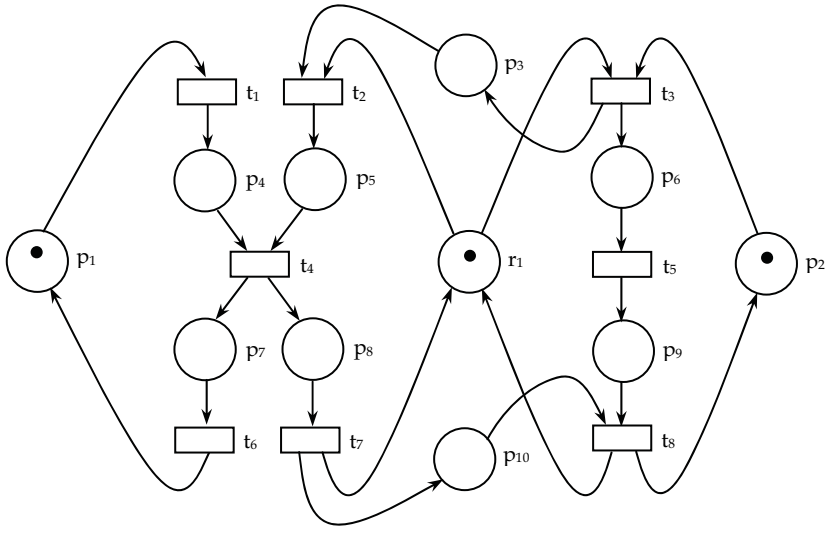


Fig. 6. An augmented marked graph for illustration of R-transform.

Property 5.3. Let (N', M_0') be the R-transform of an augmented marked graph (N, M_0; R), where r ∈ R is replaced by a set of places Q = { q_1, q_2, ..., q_k }, and P_0 be the set of marked

places in N'. Then, for each q_i in N', there exists a place invariant α_i of N' such that $\alpha_i[q_i] = 1$ and $\alpha_i[s] = 0$ for any place $s \in P_0 \setminus \{q_i\}$.

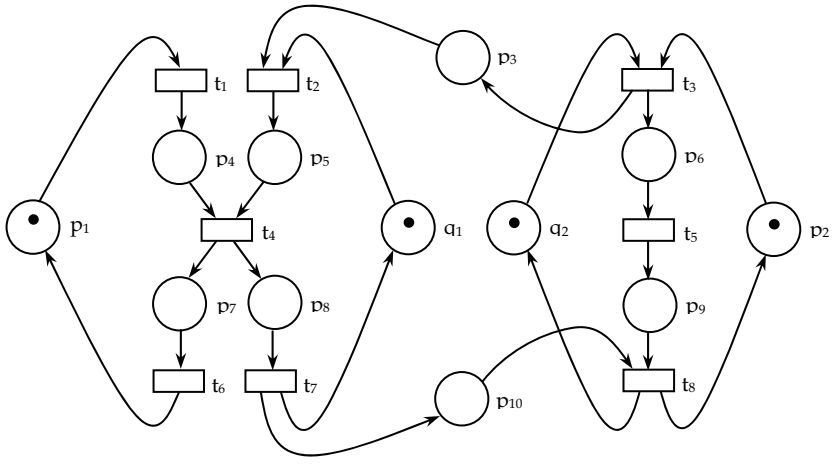


Fig. 7. The R-transform of the augmented marked graph shown in figure 6.

Proof. Let $D_r = \{ \langle t_{s1}, t_{h1}\rangle, \langle t_{s2}, t_{h2}\rangle, ..., \langle t_{skr}, t_{hkr}\rangle \}$. According to the definition of augmented marked graphs, for each $\langle t_{si}, t_{hi}\rangle$, there exists an unmarked path $\rho = \langle t_{s1}, ..., t_{h1}\rangle$ in $(N, M_0; R)$. Obviously, ρ also exists as an unmarked path in (N', M_0'), and ρ together with q_i forms a cycle γ_i which is marked at q_i only. As (N', M_0') is a marked graph, according to Property 2.5, the corresponding vector of γ_i is a place invariant α_i of N'. Since q_i is the only one marked place in γ_i, $\alpha_i[q_i] = 1$ and $\alpha_i[s] = 0$ for any place $s \in P_0 \setminus \{q_i\}$.

Property 5.4. Let $(N, M_0; R)$ be an augmented marked graph, where $R = \{ r_1, r_2, ..., r_n \}$. Let (N', M_0') be the R-transform of $(N, M_0; R)$, where each r_i is replaced by a set of places Q_i, for i = 1, 2, ..., n. If every place in (N', M_0') belongs to a cycle, then there exists a place invariant α of N' such that $\alpha > 0$ and $\alpha[q_1] = \alpha[q_2] = ... = \alpha[q_k]$ for each $Q_i = \{ q_1, q_2, ..., q_k \}$.

Proof. Let $P = \{ p_1, p_2, ..., p_n \}$ be the set of places in N', and P_0 be those marked places. Since each p_i belongs to a cycle γ_i and (N', M_0') is a marked graph, according to Property 2.5, the corresponding vector of γ_i is a place invariant α_i' of N'. Then, $\alpha' = \alpha_1' + \alpha_2' + ... + \alpha_n' > 0$ is a place invariant of N'. Consider $Q_i = \{ q_1, q_2, ..., q_k \}$. Let $q_m \in Q_i$ such that $\alpha'[q_m] \geq \alpha'[q_j]$ for any $q_j \in Q_i$. For each q_j, according to Property 5.3, there exists an invariant $\alpha_j' > 0$ such that $\alpha_j'[q_j] = 1$ and $\alpha_j'[s] = 0$ for any place $s \in P_0 \setminus \{q_j\}$. There also exists a place invariant $\alpha'' = \alpha' + h\alpha_j'$, where $h \geq 1$, such that $\alpha''[q_j] = \alpha''[q_m]$ and $\alpha''[s] = \alpha'[s]$ for any $s \in P_0 \setminus \{q_j\}$. Hence, there eventually exists a place invariant α of N' such that $\alpha[q_1] = \alpha[q_2] = ... = \alpha[q_k]$.

Consider the R-transform (N', M_0') of an augmented marked graph, as shown in Figure 7. For q_1, there exists a place invariant α_1, such that $\alpha_1[q_1] = 1$ and $\alpha_1[q_2] = \alpha_1[p_1] = \alpha_1[p_2] = 0$. For q_2, there also exists a place invariant α_2, such that $\alpha_2[q_2] = 1$ and $\alpha_2[q_1] = \alpha_2[p_1] = \alpha_2[p_2] = 0$. In (N', M_0'), every place belongs to a cycle. There also exists a place invariant $\alpha > 0$, where $\alpha[q_1] = \alpha[q_2]$.

Lemma 5.1. Let $N = \langle P, T, F \rangle$ be a PT-net and $N' = \langle P', T', F' \rangle$ be the PT-net obtained from N after fusing a set of places $Q = \{ q_1, q_2, ..., q_n \} \subset P$ into a single place $r \in P'$. If there exists a place invariant α of N such that $\alpha[q_1] = \alpha[q_2] = ... = \alpha[q_n] = k \geq 0$, then there also exists a place invariant α' of N' such that $\alpha'[r] = k$ and $\alpha'[s] = \alpha[s]$ for any $s \in P'\backslash\{r\} = P\backslash Q$.

Proof. Since N' is obtained from N by fusing $Q = \{ q_1, q_2, ..., q_n \}$ into r, we have $P' = (P\backslash Q) \cup \{ r \}$. Let V be the incidence matrix of N. Then, the incidence matrix V' of N' satisfies that $V'[r] = \Sigma_{i=1,2,...,n} V[q_i]$ and $V'[s] = V[s]$ for any $s \in P'\backslash\{r\} = P\backslash Q$. Since α is a place invariant of N, $\alpha V = 0$. Let α' be a place vector of N' such that $\alpha'[r] = \alpha[q_1] = \alpha[q_2] = ... = \alpha[q_n] = k$ and $\alpha'[s] = \alpha[s]$ for every $s \in P'\backslash\{r\} = P\backslash Q$. Then, $\alpha'V' = \alpha'[r]V'[r] + \Sigma_{p\in(P'\backslash\{r\})}\alpha'[p]V'[p] = \Sigma_{i=1,2,...,n}\alpha[q_i]V[q_i] + \Sigma_{p\in(P\backslash Q)}\alpha[p]V[p] = \alpha V = 0$. Hence, α' is a place invariant of N'.

Lemma 5.2. Let $N = \langle P, T, F \rangle$ be a PT-net and $N' = \langle P', T', F' \rangle$ be the PT-net obtained from N after fusing a set of places $Q = \{ q_1, q_2, ..., q_n \} \subset P$ into a single place $r \in P'$. If there exists a place invariant α' of N' such that $\alpha'[r] = k \geq 0$, then there also exists a place invariant α of N such that $\alpha[q_1] = \alpha[q_2] = ... = \alpha[q_n] = k$ and $\alpha[s] = \alpha'[s]$ for any $s \in P\backslash Q = P'\backslash\{r\}$.

Proof. Since N' is obtained from N by fusing $Q = \{ q_1, q_2, ..., q_n \}$ into r, we have $P' = (P\backslash Q) \cup \{ r \}$. Let V be the incidence matrix of N. Then, the incidence matrix V' of N' satisfies that $V'[r] = \Sigma_{i=1,2,...,n} V[q_i]$ and $V'[s] = V[s]$ for any $s \in P'\backslash\{r\} = P\backslash Q$. Since α' is a place invariant of N', $\alpha'V' = 0$. Let α be a place vector of N such that $\alpha[q_1] = \alpha[q_2] = ... = \alpha[q_n] = k$ and $\alpha[s] = \alpha'[s]$ for every $s \in P\backslash Q = P'\backslash\{r\}$. Then, $\alpha V = \Sigma_{i=1,2,...,n}\alpha[q_i]V[q_i] + \Sigma_{p\in(P\backslash Q)}\alpha[p]V[p] = \alpha'[r]V'[r] + \Sigma_{p\in(P'\backslash\{r\})}\alpha'[p]V'[p] = \alpha'V' = 0$. Hence, α is a place invariant of N.

Property 5.5. Let (N', M_0') be the R-transform of an augmented marked graph $(N, M_0; R)$. $(N, M_0; R)$ is bounded and conservative if and only if every place in (N', M_0') belongs to a cycle.

Proof. ($\Leftarrow$) Let $R = \{ r_1, r_2, ..., r_n \}$. (N', M_0') is the R-transform of $(N, M_0; R)$, where each r_i is replaced by a set of places Q_i, for $i = 1, 2, ..., n$. Since every place in (N', M_0') belongs to a cycle, according to Property 5.4, there exists a place invariant α' of N' such that $\alpha' > 0$ and $\alpha'[q_1] = \alpha'[q_2] = ... = \alpha'[q_k]$ for each $Q_i = \{ q_1, q_2, ..., q_k \}$. It follows from Lemma 5.1 that there also exists a place invariants α of N such that $\alpha > 0$ and $\alpha[r_i] = \alpha'[q_1] = \alpha'[q_2] = ... = \alpha'[q_k]$ for each Q_i. Hence, $(N, M_0; R)$ is conservative. According to Property 2.1, $(N, M_0; R)$ is bounded. ($\Rightarrow$) Since $(N, M_0; R)$ is conservative, there exists a place invariant α of N such that $\alpha > 0$. Consider each $r_i \in R$ which is replaced by $Q_i = \{ q_1, q_2, ..., q_k \}$. According to Lemma 5.2, there also exists a place invariant α' of N' such that $\alpha' > 0$ and $\alpha'[q_1] = \alpha'[q_2] = ... = \alpha'[q_k] = \alpha[r_i]$ and $\alpha'[s] = \alpha[s]$ for any $s \in P'\backslash Q_i$. Hence, (N', M_0') is conservative. It follows from Property 2.1 that (N', M_0') is bounded. Since (N', M_0') is a marked graph, according to Property 2.3, every place in (N', M_0') belongs to a cycle.

Property 5.6. Let (N', M_0') be the R-transform of an augmented marked graph $(N, M_0; R)$. $(N, M_0; R)$ is bounded and conservative if and only if (N', M_0') is bounded.

Proof. It follows from Properties 2.3 and 5.5.

Consider the augmented marked graph $(N, M_0; R)$ shown in Figure 6, and the R-transform (N', M_0') of $(N, M_0; R)$. Every place in (N', M_0') belongs to a cycle. $(N, M_0; R)$ is bounded and conservative. (N', M_0') is also bounded and conservative.

Figure 8 shows another augmented marked graph $(N, M_0; R)$. Figure 9 shows the R-transform (N', M_0') of $(N, M_0; R)$, where r is replaced by $\{ q_1, q_2 \}$. For (N', M_0'), places p_3 and p_{10} do not belong to any cycle. $(N, M_0; R)$ is neither bounded nor conservative.

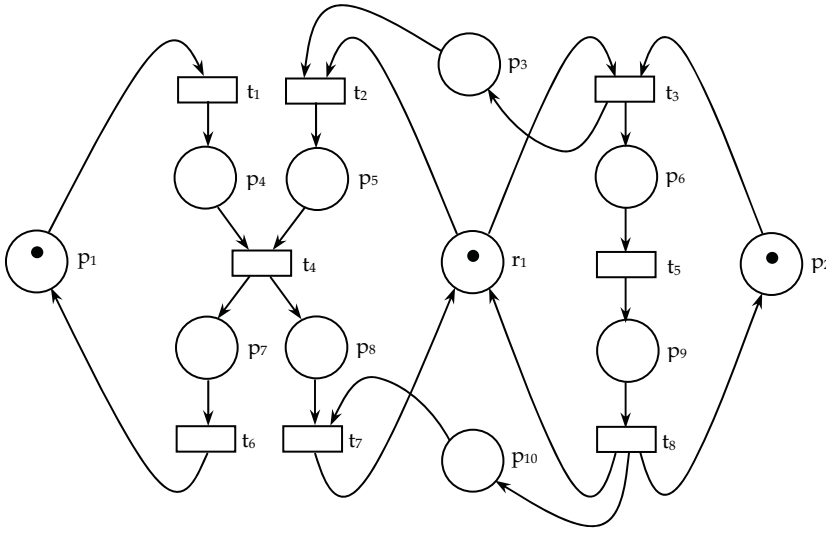


Fig. 8. Another augmented marked graph for illustration of R-transform.

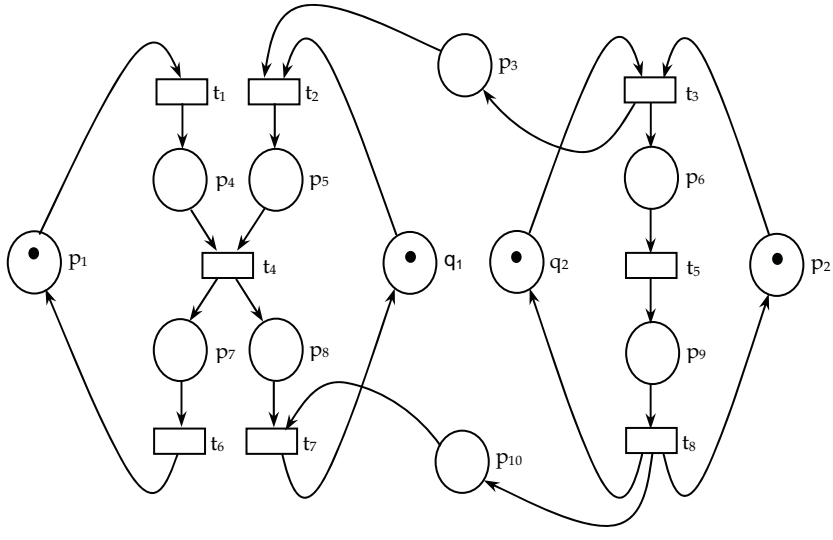


Fig. 9. The R-transform of the augmented marked graph shown in figure 8.

Based on Properties 5.5, the following strategy is derived for checking the boundedness and conservativeness of an augmented marked graph (N, M_0; R) :

1. Create the R-transform of (N, M_0; R).
2. Let (N', M_0') be the R-transform. For each place p in N', check if there exists a cycle that contains p. If yes, (N, M_0; R) is bounded and conservative. Otherwise, (N, M_0; R) is neither bounded nor conservative.

6. The Dining philosophers problem

This section illustrates the properties of augmented marked graphs obtained in the previous two sections using the dining philosophers problem. By modelling the dining philosophers problem by an augmented marked graph, the system is analysed on its liveness, reversibility, boundedness and conservativeness based on the properties of augmented marked graphs.

Example 1 : The Dining Philosophers Problem (Version 1)

Six philosophers (H_1, H_2, H_3, H_4, H_5 and H_6) are sitting around a circular table for dinner. They are either meditating or eating the food placed at the centre of the table. There are six pieces of chopsticks (C_1, C_2, C_3, C_4, C_5 and C_6) shared by them for getting the food to eat, as shown in Figure 10. For one to get the food to eat, both the chopstick at the right hand side and the chopstick at the left hand side must be available. The philosopher then grasps both chopsticks simultaneously and then takes the food to eat. Afterwards, the chopsticks are released and returned to their original positions simultaneously.

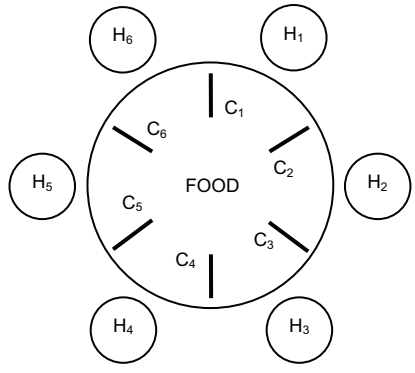


Fig. 10. The dinning philosophers problem.

Figure 11 shows an augmented marked graph (N, M_0; R), representing the dining philosophers problem (version 1). Table 1 shows the semantic meanings of the places and transitions. There are 24 R-siphons, namely, {r_1, p_{61}, p_{11}}, {r_1, p_{61}, p_{12}}, {r_1, p_{62}, p_{11}}, {r_1, p_{62}, p_{12}}, {r_2, p_{11}, p_{21}}, {r_2, p_{11}, p_{22}}, {r_2, p_{12}, p_{21}}, {r_2, p_{12}, p_{22}}, {r_3, p_{21}, p_{31}}, {r_3, p_{21}, p_{32}}, {r_3, p_{22}, p_{31}}, {r_3, p_{22}, p_{32}}, {r_4, p_{31}, p_{41}}, {r_4, p_{31}, p_{42}}, {r_4, p_{32}, p_{41}}, {r_4, p_{32}, p_{42}}, {r_5, p_{41}, p_{51}}, {r_5, p_{41}, p_{52}}, {r_5, p_{42}, p_{51}}, {r_5, p_{42}, p_{52}}, {r_6, p_{51}, p_{61}}, {r_6, p_{51}, p_{62}}, {r_6, p_{52}, p_{61}} and {r_6, p_{52}, p_{62}}. Each of these R-siphons contains a marked trap and would never become empty. Based on the results obtained in Section 4, (N, M_0; R) is live and reversible. On the other hand, for the R-transform of (N, M_0; R), every place belongs to a cycle. Based on the results obtained in Section 5, (N, M_0; R) is bounded and conservative.

Example 2 : The Dining Philosophers Problem (Version 2)

The Dining Philosophers Problem is now modified as follows. For one to get the food to eat, he or she first grasps the chopstick at the right hand side if available, then grasps the chopstick at the left hand side if available, and then takes the food to eat. Afterwards, the chopsticks are released and returned to their original positions simultaneously.

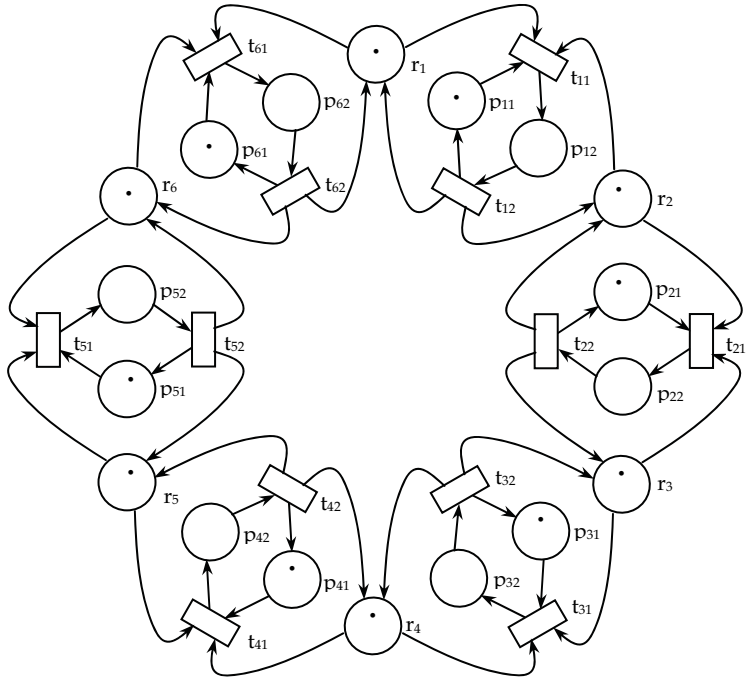


Fig. 11. Augmented marked graph (Example 1).

Semantic meaning for places		Semantic meaning for transitions	
p_{11}	H_1 is meditating.	t_{11}	H_1 takes the action to grasp C_1 and C_2.
p_{12}	H_1 has got C_1 and C_2 and takes the food.	t_{12}	H_1 takes the action to return C_1 and C_2.
p_{21}	H_2 is meditating.	t_{21}	H_1 takes the action to grasp C_2 and C_3.
p_{22}	H_2 has got C_2 and C_3 and takes the food.	t_{22}	H_1 takes the action to return C_2 and C_3.
p_{31}	H_3 is meditating.	t_{31}	H_1 takes the action to grasp C_3 and C_4.
p_{32}	H_3 has got C_3 and C_4 and takes the food.	t_{32}	H_1 takes the action to return C_3 and C_4.
p_{41}	H_4 is meditating.	t_{41}	H_1 takes the action to grasp C_4 and C_5.
p_{42}	H_4 has got C_4 and C_5 and takes the food.	t_{42}	H_1 takes the action to return C_4 and C_5.
p_{51}	H_5 is meditating.	t_{51}	H_1 takes the action to grasp C_5 and C_6.
p_{52}	H_5 has got C_5 and C_6 and takes the food.	t_{52}	H_1 takes the action to return C_5 and C_6.
p_{61}	H_6 is meditating.	t_{61}	H_1 takes the action to grasp C_6 and C_1.
p_{62}	H_6 has got C_6 and C_1 and takes the food.	t_{62}	H_1 takes the action to return C_6 and C_1.
r_1	C_1 is available for pick.		
r_2	C_2 is available for pick.		
r_3	C_3 is available for pick.		
r_4	C_4 is available for pick.		
r_5	C_5 is available for pick.		
r_6	C_6 is available for pick.		

Table 1. Semantic meaning for the places and transitions in Fig. 11.

Figure 12 shows an augmented marked graph (N, M_0; R), representing the dining philosophers problem (version 2). Table 2 shows the semantic meanings of the places and transitions. The set of places {r_1, p_{13}, r_2, p_{23}, r_3, p_{33}, r_4, p_{43}, r_5, p_{53}, r_6, p_{63}} is a R-siphon which would become empty after firing the sequence of transitions ⟨t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}⟩. Based on the results obtained in Section 4, (N, M_0; R) is neither live nor reversible. Deadlocks would occur, for example, after firing ⟨t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}⟩. On the other hand, for the R-transform of (N, M_0; R), every place belongs to a cycle. Based on the results obtained in Section 5, (N, M_0; R) is bounded and conservative.

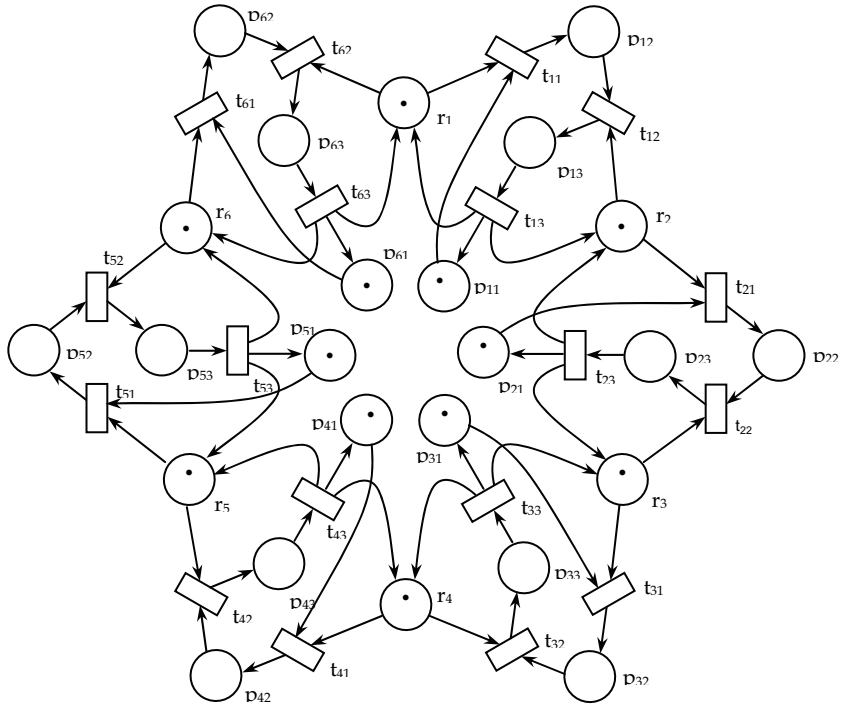


Fig. 12. Augmented marked graph (Example 2).

7. Application to manufacturing system

Manufacturing systems are typically shared resource systems, wherein the resources used to be maximally shared among different asynchronous processes. Moreover, every resource has a pre-defined capacity limit that can never be exceeded. Therefore, in manufacturing system design, a major design objective is to achieve a live, bounded and reversible system - liveness implies freeness of deadlock, boundedness implies absence of capacity overflow, and reversibility allows system recovery. Verification of the system liveness, boundedness and reversibility is essentially required, though very time-consuming.

Possessing a specific structure for representing shared resources, augmented marked graphs are often used for modelling manufacturing systems. By modelling a manufacturing system as an augmented marked graph, this section shows how the system liveness, boundedness and reversibility can be analysed, based on the properties of augmented marked graphs.

Semantic meaning for places		Semantic meaning for transitions	
p_{11}	H_1 is meditating.	t_{11}	H_1 takes the action to grasp C_1.
p_{12}	H_1 has got C_1 and prepares to pick C_2.	t_{12}	H_1 takes the action to grasp C_2.
p_{13}	H_1 has got C_1 and C_2 and takes the food.	t_{13}	H_1 takes the action to return C_1 and C_2.
p_{21}	H_2 is meditating.	t_{21}	H_2 takes the action to grasp C_2.
p_{22}	H_2 has got C_2 and prepares to pick C_3.	t_{22}	H_2 takes the action to grasp C_3.
p_{23}	H_2 has got C_2 and C_3 and takes the food.	t_{23}	H_2 takes the action to return C_2 and C_3.
p_{31}	H_3 is meditating.	t_{31}	H_3 takes the action to grasp C_3.
p_{32}	H_3 has got C_3 and prepares to pick C_4.	t_{32}	H_3 takes the action to grasp C_4.
p_{33}	H_3 has got C_3 and C_4 and takes the food.	t_{33}	H_3 takes the action to return C_3 and C_4.
p_{41}	H_4 is meditating.	t_{41}	H_4 takes the action to grasp C_4.
p_{42}	H_4 has got C_4 and prepares to pick C_5.	t_{42}	H_4 takes the action to grasp C_5.
p_{43}	H_4 has got C_4 and C_5 and takes the food.	t_{43}	H_4 takes the action to return C_4 and C_5.
p_{51}	H_5 is meditating.	t_{51}	H_5 takes the action to grasp C_5.
p_{52}	H_5 has got C_5 and prepares to pick C_6.	t_{52}	H_5 takes the action to grasp C_6.
p_{53}	H_5 has got C_5 and C_6 and takes the food.	t_{53}	H_5 takes the action to return C_5 and C_6.
p_{61}	H_6 is meditating.	t_{61}	H_6 takes the action to grasp C_6.
p_{62}	H_6 has got C_6 and prepares to pick C_1.	t_{62}	H_6 takes the action to grasp C_1.
p_{63}	H_6 has got C_6 and C_1 and takes the food.	t_{63}	H_6 takes the action to return C_6 and C_1.
r_1	C_1 is available for pick.		
r_2	C_2 is available for pick.		
r_3	C_3 is available for pick.		
r_4	C_4 is available for pick.		
r_5	C_5 is available for pick.		
r_6	C_6 is available for pick.		

Table 2. Semantic meaning for the places and transitions in Fig. 12.

Example 3. It is a FWS-200 Flexible Workstation System, extracted from the literature (Zhou & Venkatesh, 1999, pp. 121-124). The system consists of two robots R_1 and R_2, one feeder area and one PCB area, as shown in Figure 13. There are two asynchronous processes.

Production process 1 : R_1 picks components from the feeder area, and moves into the PCB area for inserting components. The finished product is then moved out from the PCB area.
Production process 2 : R_2 picks components from the feeder area, and moves into the PCB area for inserting components. The finished product is then moved out from the PCB area.
Figure 14 shows an augmented marked graph (N, M_0; R), representing the FWS-200 flexible workstation system. Table 3 shows the semantic meanings of the places and transitions.

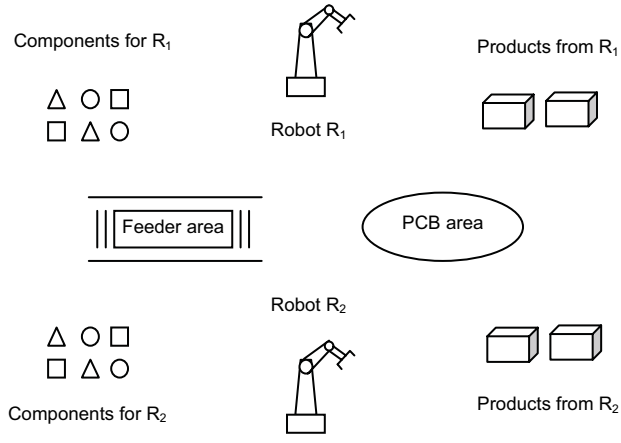


Fig. 13. The FWS-200 flexible workstation system.

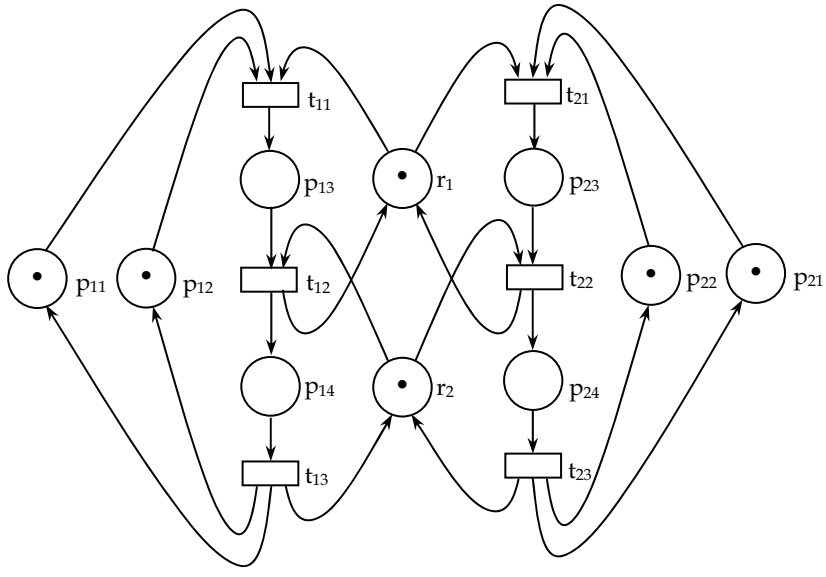


Fig. 14. Augmented marked graph (Example 3).

For the augmented marked graph (N, M_0; R) shown in Figure 14, every R-siphon would never become empty. Based on the results obtained in Section 4, (N, M_0; R) is live and

reversible. For the R-transform of (N, M_0; R), every place belongs to a cycle. Based on the results obtained in Section 5, (N, M_0; R) is bounded and conservative. It is then concluded that the FWS-200 flexible workstation system is live, bounded, reversible and conservative.

Semantic meaning for places		Semantic meaning for transitions	
p_{11}	R_1 is ready	t_{11}	R_1 starts picking components
p_{12}	Components for R_1 are available	t_{12}	R_1 starts inserting components
p_{13}	R_1 is picking components from feeder	t_{13}	R_1 starts moving out the product
p_{14}	R_1 is inserting components in PCB area	t_{21}	R_2 starts picking components
p_{21}	R_2 is ready	t_{22}	R_2 starts inserting components
p_{22}	Components for R_2 are available	t_{23}	R_2 starts moving out the finished product
p_{23}	R_2 is picking components from feeder		
p_{24}	R_2 is inserting components in PCB area		
r_1	Feeder area is available		
r_2	PCB area is available		

Table 3. Semantic meaning for the places and transitions in Fig. 14.

Example 4. It is a flexible assembly system, extracted from the literature (Proth & Xie, 1996, pp. 58-61). The system consists of three conveyors C_1, C_2 and C_3 and three robots R_1, R_2 and R_3, as shown in Figure 15. There are three asynchronous processes.
Assembly process 1 : C_1 requests R_1. After acquiring R_1, it requests R_2. After acquiring R_2, it performs assembling and then releases both R_1 and R_2 simultaneously.
Assembly process 2 : C_2 requests R_2. After acquiring R_2, it requests R_3. After acquiring R_3, it perform assembling and then releases both R_2 and R_3 simultaneously.
Assembly process 3 : C_3 requests R_3. After acquiring R_3, it requests R_1. After acquiring R_1, it perform assembling and then releases both R_3 and R_1 simultaneously.

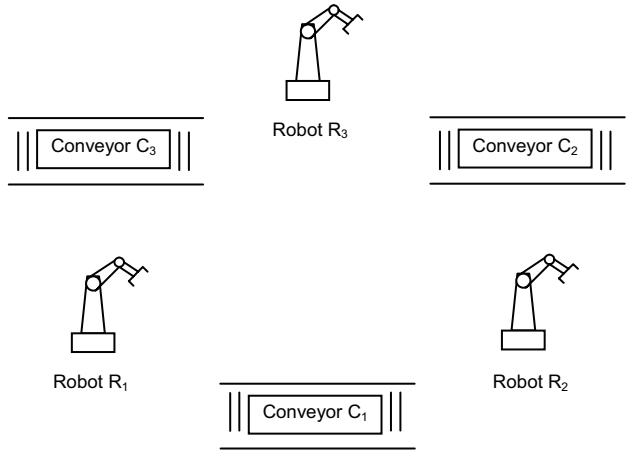


Fig. 15. The flexible assembly system.

Figure 16 shows an augmented marked graph (N, M_0; R), representing the flexible assembly system. Table 4 shows the semantic meanings of the places and transitions.

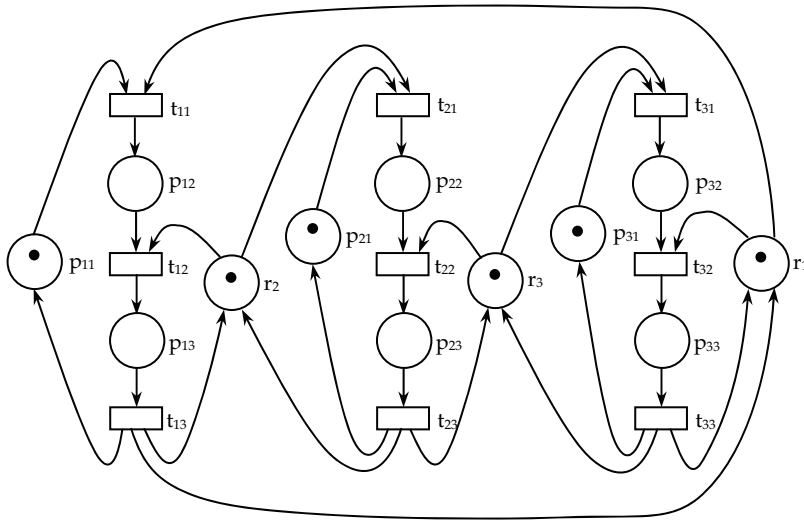


Fig. 16. Augmented marked graph (Example 4).

Semantic meaning for places		Semantic meaning for transitions	
p_{12}	C_1 is occupying R_1	t_{11}	C_1 starts acquiring R_1
p_{13}	C_1 is occupying R_1 and R_2	t_{12}	C_1 starts acquiring R_2
p_{21}	C_2 is ready	t_{13}	C_1 finishes assembling and release R_1 and R_2
p_{22}	C_2 is occupying R_2	t_{21}	C_2 starts acquiring R_2
p_{23}	C_2 is occupying R_2 and R_3	t_{22}	C_2 starts acquiring R_3
p_{21}	C_3 is ready	t_{23}	C_2 finishes assembling and release R_2 and R_3
p_{22}	C_3 is occupying R_3	t_{31}	C_3 starts acquiring R_3
p_{23}	C_3 is occupying R_3 and R_1	t_{32}	C_3 starts acquiring R_1
r_1	R_1 is available	t_{33}	C_3 finishes assembling and release R_3 and R_1
r_2	R_2 is available		
r_3	R_3 is available		

Table 4. Semantic meaning for the places and transitions in Fig. 16.

For the augmented marked graph (N, M_0; R) shown in Figure 16, there exists a R-siphon S = { p_{13}, p_{23}, p_{33}, r_1, r_2, r_3 } which becomes empty after firing the sequence of transitions ⟨ t_{11}, t_{21}, t_{31} ⟩. Based on the results obtained in Section 4, (N, M_0; R) is neither live nor reversible. A deadlock would occur after firing ⟨ t_{11}, t_{21}, t_{31} ⟩. For the R-transform of (N, M_0; R), every place belongs to a cycle. Based on the results obtained in Section 5, (N, M_0; R) is bounded and conservative. It is then concluded that the flexible assembly system is non-live, non-reversible but bounded and conservative.

8. Conclusion

In the past decade, augmented marked graphs have evolved into a sub-class of Petri nets. They are often used for modelling shared resource systems, such as manufacturing systems. One major reason is that augmented marked graphs possess a special structure which is desirable for modelling shared resources. However, the properties of augmented marked graphs are not extensively studied. In the literature, there are a few published works on augmented marked graphs.

This paper consolidates our earlier works on augmented marked graphs with a special focus on liveness, boundedness, reversibility and conservativeness. We provide a number of characterisations for live and reversible augmented marked graphs. In particulars, some of these characterisations are based on cycles, instead of siphons. Besides, we introduce the R-transformation, on which characterisations for bounded and conservative augmented marked graphs are obtained. With these characterisations, some pretty simple conditions and procedures for checking the liveness, reversibility, boundedness and conservativeness of an augmented marked graph are derived. These have been illustrated using the dining philosophers problem.

Typically, in designing shared resource systems, one need to achieve design objectives on two folds. On one hand, the resources are scarce and should be maximally shared. On the other hand, the system should be carefully designed so that erroneous situations due to the sharing of resources, such as deadlock and capacity overflow, can be avoided. Yet, the verification of liveness, boundedness and reversibility is very difficult and time-consuming. This paper contributes to provide an effective means to analysing these essential properties. By modelling a shared resource system as an augmented marked graph, its liveness, boundedness, reversibility and conservativeness can be effectively analysed, based on the characterisations and properties of augmented marked graphs. We specifically show the application to the analysis of manufacturing systems which are typically shared resource systems. Promising results are obtained.

9. References

Barkaoui, K., Couvreur, J.M. & Dutheillet, C. (1995), On Liveness in Extended Non Self-Controlling Nets, *Application and Theory of Petri Nets, Lecture Notes in Computer Science*, Vol. 935, pp. 25-44, Springer-Verlag.

Cheung, K.S. (2004), New Characterisations for Live and Reversible Augmented Marked Graphs, *Information Processing Letters*, Vol. 92, No. 5, pp. 239-243.

Cheung, K.S. (2005), A Synthesis Method for Designing Shared-Resource Systems, *Computing and Informatics*, Vol. 24, No. 6, pp. 629-653.

Cheung, K.S. (2006), Modelling and Analysis of Manufacturing Systems Using Augmented Marked Graphs, *Information Technology and Control*, Vol. 35, No. 1, pp. 19-26.

Cheung, K.S. (2007), Boundedness and Conservativeness of Augmented Marked Graphs, *IMA Journal of Mathematical Control and Information*, Vol. 24, No. 2, pp. 235-244.

Cheung, K.S. & Chow, K.O. (2005a), Cycle-Inclusion Property of Augmented Marked Graphs, *Information Processing Letters*, Vol. 94, No. 6, pp. 271-276.

Cheung, K.S. & Chow, K.O. (2005b), Analysis of Manufacturing Systems Based on Augmented Marked Graphs, *Proceedings of the IEEE International Conference on Computational Intelligence for Modelling, Control and Automation*, Vol. 2, pp. 847-851.

Cheung, K.S. & Chow, K.O. (2005c), A Synthesis Approach to Deriving Object-Based Specifications from Object Interaction Scenarios, In : Nilsson et al. (eds.), *Advances in Information Systems Development - Bridging the Gap Between Academic and Industry*, pp. 647-656, Springer.

Cheung, K.S. & Chow, K.O. (2006), Analysis of Capacity Overflow for Manufacturing Systems, *Proceedings of the IEEE Conference on Automation Science and Engineering*, pp. 287-292.

Cheung, K.S., Cheung, T.Y. & Chow, K.O. (2006), A Petri-Net-Based Synthesis Methodology for Use-Case-Driven System Design, *Journal of Systems and Software*, Vol. 79, No. 6, pp. 772-790.

Chu, F. & Xie, X. (1997), Deadlock Analysis of Petri Nets Using Siphons and Mathematical Programming, *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 6, pp. 793-804.

Desel, J. & Esparza, J. (1995), *Free Choice Petri Nets*, Cambridge University Press.

Desel, J. & Reisig, W. (1998), Place Transition Petri Nets, *Lectures on Petri Nets I : Basic Models, Lecture Notes in Computer Science*, Vol. 1491, pp. 122-173, Springer-Verlag.

Huang, H.J., Jiao, L. & Cheung, T.Y. (2003), Property-Preserving Composition of Augmented Marked Graphs that Share Common Resources, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1446-1451.

Jeng, M.D., Xie, X. & Huang, Y. (2000), Manufacturing Modeling Using Process Nets with Resources, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2185-2190.

Jeng, M.D., Xie, X. & Peng, M.Y. (2002), Process Nets with Resources for Manufacturing Modeling and their Analysis, *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 6, pp. 875-889.

Murata, T. (1989), Petri Nets : Properties, Analysis and Applications, *Proceedings of the IEEE*, Vol. 77, No. 4., pp. 541-580.

Peterson, J.L. (1981), *Petri net theory and the modeling of systems*, Prentice Hall.

Proth, J.M. & Xie, X. (1996), *Petri Nets : A Tool for Design and Management of Manufacturing Systems*, Wiley.

Reisig, W. (1985), *Petri Nets : An Introduction*, Springer-Verlag.

Zhou, M.C. & Venkatesh, K. (1999), *Modeling, Simulation and Control of Flexible Manufacturing Systems : A Petri Net Approach*, World Scientific.

18

Incremental Integer Linear Programming Models for Petri Nets Reachability Problems

Thomas Bourdeaud'huy[1], Saïd Hanafi[2] and Pascal Yim[1]
[1]*L.A.G.I.S. Ecole Centrale de Lille*
[2]*L.A.M.I.H. Université de Valenciennes*
France

1. Introduction

The operational management of complex systems is characterized, in general, by the existence of a huge number of solutions. Decision-making processes must be implemented in order to find the best results. These processes need suitable modeling tools offering true practical resolution perspectives. Among them, Petri nets (PNs) provide a simple graphical model taking into account, in the same formalism, concurrency, parallelism and synchronization. Their graphical and precise nature, their firm mathematical foundation and the aboundance of analysis methods have made them become a classical modeling tool for the study of discrete event systems, ranging from operating systems to logistic ones. However, their interest in the field of problem solving is still badly known.

In this paper, we consider some PN *reachability problems*. Since PNs can model flows in a natural and efficient way, many operations research problems can be defined using reachability between states, e.g. scheduling (Lee and DiCesare, 1994; Van Der AAlst, 1995), planning (Silva et al., 2000), car-sequencing problems (Briand, 1999). Moreover, research on Petri nets addresses the issue of flexibility: many extensions have been proposed to facilitate the modeling of complex systems, by addition of ``color'', ``time'' and ``hierarchy'' (Jensen, 1992; Wang,1998). For example, it is relatively easy to map scheduling problems onto timed PNs. Their graphical nature reinforce obviously this strength, by allowing a kind of *interactivity* with the system. At last, a large number of difficult PN analysis problems are equivalent to the reachability problem, or to some of its variants or sub-problems (Keller, 1976). Particularly, *model-checking* (Latvala, 2001) which represents a key point when dealing with systems analysis is *directly* linked to an exhaustive traversal of the corresponding PN reachability graph.

Various methods have been suggested to handle the PNs reachability problem. In this paper, we propose to use the mathematical programming paradigm. Some PN analysis problems have already been handled using such techniques (Melzer and Esparza, 1996;Silva et al., 1998; Khomenko and Koutny, 2000), but none has considered the general PNs reachability problem.

The proposed approach is based on an *implicit traversal* of the Petri net reachability graph, which does not need its construction. This is done by considering a unique sequence of *steps* growing incrementally to represent exactly the total behavior of the net. We follow here a

previous work from (Benasser and Yim, 1999) called *logical abstraction technique*. Their technique was validated on several examples using logical constraint programming techniques. It has shown more effective than other generic solvers and could even compete with heuristics dedicated to particular classes of problems. Our methodology allows to improve this original model using the wide range of tools and adjustments brought by *Operational Research* techniques. We model the problem as an integer linear program, then we solve it with a branch-and-bound technique (divide and conquer), using the Cplex optimization software.
Moreover, we show how our incremental approach can be extended to *Timed Petri nets* in order to solve scheduling problems modelled as Timed Petri Nets reachability problems. The model built is as general as possible since we do not make assumptions about the firing policy, contrarywise to other classical approaches dealing with the same issue.
This chapter is organized as follows. In section 2, we formally define the kind of PN considered, their respective reachabilty problems and the ways such problems are dealt with in the litterature. Then, in section 3, we give general considerations about step firings and describe the elements of our incremental approaches. In section 4, we apply our methodology to express reachability problems using a mathematical programming formulation. Finally, as a conclusion, we describe a few promising research directions.

2. Petri Nets reachability problems

In this section, we give the terminology of both kinds of the PN we are interested in using linear algebra -- in order to make our formulations more concise -- and define formally their respective reachability problems.

2.1 Place/transition Petri nets

2.1.1 Petri net terminology

Definition 1 (Place/Transition Petri Net). *A Place/Transition Petri net (Murata, 1989)* $R = (\mathrm{P}, \mathrm{T}, C^-, C^+)$ *with its initial marking* m *is a bipartite weighted directed graph where:*

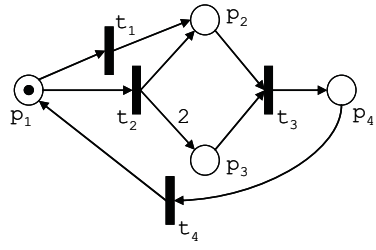
- $\mathrm{P} = \{p_1, \ldots, p_m\}$ *is a finite set of places, with* $M = |\mathrm{P}|$ *. Places are represented as circles and indexed by letter* i *;*
- $\mathrm{T} = \{t_1, \ldots, t_n\}$ *is a finite set of transitions, with* $N = |\mathrm{T}|$ *. Transitions are represented as rectangles and indexed by letter* j *;*
- *Incidence matrices* C^-, C^+ *and* $C \in \mathrm{N}^{\mathrm{P}\times\mathrm{T}}$ *(with* $C = C^+ - C^-$*) define the weighted flow function which associates to each arc* (p_i, t_j) *(from place* p_i *to transition* t_j*) or* (t_j, p_i) *(from transition* t_j *to place* p_i*) its weight* C_{ij}^- *or* C_{ij}^+*. When there is no arc between place* p_i *and transition* t_j*, then we have:* $C_{ij}^- = C_{ij}^+ = 0$*. The* i^{th} *row vector and* j^{th} *column vector taken from incidence matrices* C^-*,* C^+ *and* C *are denoted respectively* C_i^-*,* C_j^-*,* C_i^+*,* C_j^+ *and* C_i*,* C_j*. We denote respectively by* ${}^\bullet p$ *and* $p^\bullet$ *the set of predecessors and*

successors of place p*, and conversely* ${}^{\bullet}t$ *and* $t^{\bullet}$ *are the set of predecessors and successors of transition* t *(also known as input and output nodes);*

- $m : \mathrm{P} \to \mathrm{N}$ *associates to each place* $p \in \mathrm{P}$ *an integer* $m(p)$ *called the marking of the place* p*. Markings are represented as full dots called tokens inside places.*

Definition 2 (Characteristic Vectors). *Let* (R, m) *be a Petri net with* $\mathrm{P} = \{p_1, p_2, \ldots, p_m\}$ *and* $\mathrm{T} = \{t_1, t_2, \ldots, t_n\}$*:*

- *The canonical vector* $\overrightarrow{e_{p_i}}$ *associated to place* p_i *(resp.* $\overrightarrow{e_{t_j}}$ *associated to transition* t_j*) is the vector in* $\{0,1\}^N$ *(resp. in* $\{0,1\}^M$*) which takes the value ``1'' in its* $i^{th}$ *(resp.* $j^{th}$*) component and ``0'' elsewhere.*
- *The marking vector* $\overrightarrow{m}$ *associated to marking* m *is the column vector* $(m(p_1), m(p_2), \ldots, m(p_m))^{\acute{u}} \in \mathrm{N}^M$*.*



$$C^- = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad C^+ = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad C = \begin{bmatrix} -1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Fig. 1. A Petri Net and its Incidence Matrices

Example 1 (PN). *An example of a PN and its incidence matrices is presented in Fig.1. Its initial marking is* $m_0 = (1,0,0,0)^{\acute{u}}$*. We have* $\overrightarrow{e_{p_2}} = (0,1,0,0,0)^{\acute{u}}$ *and* $\overrightarrow{e_{t_3}} = (0,0,1,0,0)^{\acute{u}}$*.*

In a PN, the markings of the places represent the state of the corresponding system at a given moment. This state can be modified by the *firing* of transitions. This behaviour is called the ``*token game*''.

Definition 3 (Transition Firings). *Let* (R, m) *be a Petri net. A transition* t_j *is fireable from marking* m *iff:*

$$\begin{array}{rlcl} & \forall p_i \in \mathrm{P}, & m(p_i) & \geq & C^-_{ij} \\ \Leftrightarrow & & \overrightarrow{m} & \geq & C^- \cdot \overrightarrow{e_{t_j}} \end{array} \tag{1}$$

The fireability condition is denoted by $m[t\rangle$*. If this condition is satisfied, a new marking* m' *is produced from the marking* m*, such that:*

$$\begin{array}{rrcl} & \forall p_i \in \mathrm{P}, \quad m'(p_i) & = & m(p_i) - C_{ij}^{-} + C_{ij}^{+} \\ \Leftrightarrow & \overrightarrow{m'} & = & \overrightarrow{m} + C \cdot \overrightarrow{e_{t_j}} \end{array} \tag{2}$$

The firing of a transition t *from the marking* m *to the marking* m' *is denoted by* $m[t\rangle m'$.
Transition firings modify the marking of the net. It is thus interesting to know if one particular marking can be reached. This problem is known as the ``*reachability problem*'' for Petri nets.

2.1.2 Reachability problem

Definition 4 (Reachable Marking). *A marking* m' *is reachable from a marking* m *iff there exists a sequence of transitions* $\sigma = t_{\sigma_1} t_{\sigma_2} \ldots t_{\sigma_k}$ *such that:* $m[t_{\sigma_1}\rangle m_1[t_{\sigma_2}\rangle m_2 \ldots [t_{\sigma_K}\rangle m'$

We denote by $m[\sigma\rangle m'$ that the marking m' is reachable from the marking m, where $\sigma = t_{\sigma_1} t_{\sigma_2} \ldots t_{\sigma_K}$ is called a *firing sequence.* The *Parikh vector* $\vec{\sigma} = \sum_{k=1}^{K} \overrightarrow{e_{t_{\sigma_k}}}$ associated to the firing sequence σ is the vector whose j^{th} component is equal to the number of times the transition j is fired in σ. It is used to formulate a well known property of Petri Nets.

Proposition 1 (State equation). *Let* (R, m_0) *be a Petri net,* m_f *a marking and* $\sigma = t_{\sigma_1} t_{\sigma_2} \ldots t_{\sigma_K}$ *a firing sequence. Then we have:*

$$m_0[\sigma\rangle m_f \Rightarrow \overrightarrow{m_f} = \overrightarrow{m_0} + C \cdot \vec{\sigma} \tag{3}$$

Proof. It is obtained using a simple induction over the number of transitions fired in the sequence. W

The equation **Error! Reference source not found.** is called the *fundamental (or state) equation* of Petri nets. This equation has been widely studied in PN reachability analysis, but it only leads to semi-decision algorithms due to the existence of *spurious solutions* (Silva et al., 1992). Indeed, in that case, the reverse implication does not hold: the Parikh vector of a firing sequence is always solution to the state equation, but the reverse is not true. Some techniques (Colom and Silva, 1989b) have been proposed to improve the strength of this characterization, but they are still insufficient.

Definition 5 (Reachability Problem). *Let* (R, m_0) *be a Petri net and* m_f *a marking. The set of all markings reachable from* m_0 *is denoted by* $\mathrm{R}(R, m_0)$*; the set of all possible firing sequences (within which each transition is fireable from the corresponding marking) is denoted by* $\mathrm{F}(R, m_0)$*. The problem of finding whether* $m_f \in \mathrm{R}(R, m_0)$ *or not is known as the reachability problem for Petri nets.*

It has been shown that the reachability problem is decidable (Kosaraju, 1982). However it is EXP-TIME and EXP-SPACE hard in the general case (Lipton, 1976). Of course, practical

applications need not only to know if a marking is reachable, but also what are the corresponding firing sequences leading to this marking. To solve this problem, one needs to find a firing sequence $\sigma \in \mathrm{F}(R, m_0)$ such that $m_0[\sigma\rangle m_f$. A ``naive'' approach consists in exploring the *reachability graph* exhaustively. This graph corresponds to the usual formal representation of the behavior of the net.

Definition 6 (Reachability Graph). *The reachability graph of a Petri net* (R, m_0)*, denoted by* $\mathrm{G}(R, m_0)$*, is defined by:*

A set of nodes $\mathrm{R}(R, m_0)$ which represents the reachable markings;

A set of arcs, where an arc (m, m') labelled t connects nodes m and m' iff $m[t\rangle m'$.

Example 2 (Reachability Graph). *Fig.2. presents a part of the reachability graph for the Petri net of Fig.1.*

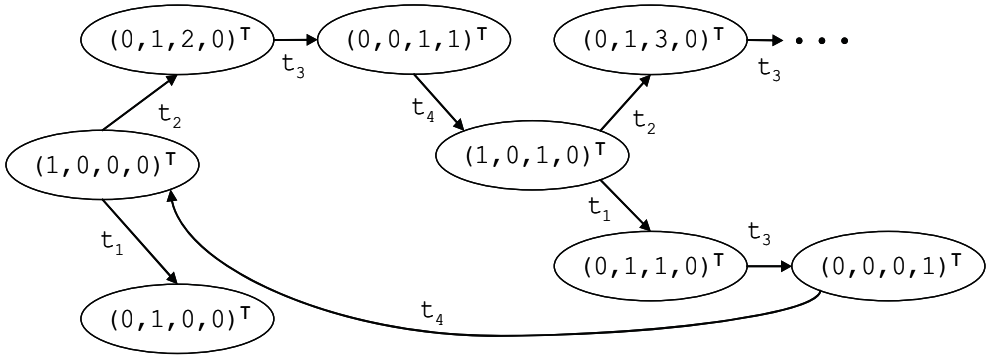


Fig. 2. Reachability graph for the PN of Fig. 1

For a given initial marking m_0, the reachability graph $\mathrm{G}(R, m_0)$ and the corresponding reachability set $\mathrm{R}(R, m_0)$ may be of infinite size. For instance, the set of markings reachable from m_0 for the net of Fig. 1 is infinite.

Practically, it is not possible to explore the reachability graph exhaustively due to the well known problem of *combinatorial explosion*: the size of the state-space (i.e. the size of the reachability set) may grow exponentially with the size of a system configuration (i.e. the number of nodes of the Petri net). Many methods have been studied to limit this explosion. Let us mention the three main families.

First ones aims at *managing* the combinatorial explosion without modifying the studied reachability graph. Classical approaches are *graph compressions*, particularly *bdd encoding* (Gunnarsson, 1998) and *forward checking* (Fernandez et al., 1992). Both uses depth first traversal of the reachability graph.

- Other techniques construct a *reduced* reachability graph associated to the original, based on some properties to preserve: symmetries (Huber et al., 1985), reductions (Berthelot, 1986) and partial order (*covering step graphs* (Vernadat et al., 1996), *stubborn sets*

(Valmari, 1991)) are the main approaches. The *logical abstraction technique* (Benasser and Yim, 1999) belongs also to this category.

- Last ones are based on the PN state equation (cf. Proposition 1): we can distinguish *parametrized analysis* (Lindqvist, 1993) and *algebraic methods* (Lautenbach, 1987).

Many extensions have been proposed to improve the modelling power of Petri nets. Among them, several extended Petri nets with ``time'' have been proposed by assigning punctual firing times (leading to ``*Timed PN*'') or time intervals (``*Time PN*'') to the components of Petri nets (transitions, places, arcs or even tokens). To deal with firing times, two main methods for modeling timing are used: either the timings are associated with the places (the PN is said to be P-timed) (Sifakis, 1975), or the timings are associated with the transitions (the PN is said to be T-timed) (Ramchandani, 1974). Depending on the system to be modeled, one of the models (P-timed or T-timed) may be easier to use than the other one. However, Sifakis has shown that the two models are equivalent. In the context of scheduling problems, (Hillion and Proth, 1989) and (Van Der Aalst, 1995) propose to use T-timed Petri nets, hereafter called simply *Timed PN*. We describe this model in the following section.

2.2 Timed Petri nets

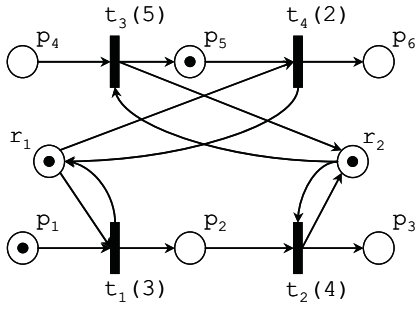
Timed Petri nets have been introduced by (Ramchandani, 1974). The following presentation has been adapted from (Chrétienne, 1984). We start by giving an informal introduction on Timed Petri nets.

2.2.1 Informal presentation

Timed Petri nets correspond to Places/Transitions Petri nets where a *duration* $d(t) \in \mathrm{N}^*$ is associated to each transition t. A Timed Petri net has the same representation as PN, to which is added a *labelling* on transitions. An example of Timed Petri net is given in Fig. 3. We have: $d(t_1) = 3$, $d(t_2) = 4$, $d(t_3) = 5$, $d(t_4) = 2$.

The firing durations associated to transitions modify the *marking validity conditions*. As soon as durations are associated to transitions, the Petri net acts as if tokens ``*disappeared*'' at the time the transition is fired, and then ``*reappeared*'' after a delay corresponding to the duration of the fired transition. Thus, the marking of a Timed Petri net evolves with the occurences of an external timer. For instance, let's consider the Timed Petri net of Fig. 3. At date 1, the transition t_1 (duration: *3 t.u.*) is fired. Then the transition t_4 (duration: *2 t.u.*) is fired at date 5. The evolution of marking with time is given in Fig. 3. Note that one could have fired transition t_4 at date 4, since the resource r_1 had been released at the end of the firing of transition t_1. However, the same transition was not fireable at date 3, since the firing of t_1 was not finished.

The firing and ending dates of transitions play a fundamental role in the behaviour of the Timed Petri net. It is thus necessary to *adapt* the firing equations according to these firing dates. In order to respect the underlying semantic of PN, a *timed firing sequence* is said to be *feasible* if and only if, at any time, the transient marking reached is made of *non negative* components.



	Date	Marking $(p_1,p_2,p_3,p_4,p_5,p_6,r_1,r_2)^{\acute{u}}$
Initial date	0	$(1,0,0,0,1,0,1,1)^{\acute{u}}$
Firing of $t_1 \rightarrow$	1	$(0,0,0,0,1,0,0,1)^{\acute{u}}$
	2	$(0,0,0,0,1,0,0,1)^{\acute{u}}$
	3	$(0,0,0,0,1,0,0,1)^{\acute{u}}$
End of $t_1 \rightarrow$	4	$(0,1,0,0,1,0,1,1)^{\acute{u}}$
Firing of $t_4 \rightarrow$	5	$(0,1,0,0,0,0,0,1)^{\acute{u}}$
	6	$(0,1,0,0,0,0,0,1)^{\acute{u}}$
End of $t_4 \rightarrow$	7	$(0,1,0,0,0,1,1,1)^{\acute{u}}$
	8	$(0,1,0,0,0,1,1,1)^{\acute{u}}$

Fig. 3. Example of a Timed Petri Net and a Timed Firing Sequence

2.2.2 Timed Petri nets terminology

Definition 7 (TPN -- Timed Petri Net). *A Timed Petri net (Ramchandani, 1974) is defined by a pair* (R,d) *where* R *is a Place/Transition Petri Net and* $d:\mathrm{T}\rightarrow\mathrm{N}^*$ *is a mapping associating a duration to each transition of the net. The vector* $\vec{d}=\sum_{t\in\mathrm{T}} d(t)\cdot\vec{e_t}$ *is called the duration vector of the Timed Petri net.*

Note that one could more generally consider *rational valued* durations. Nevertheless, after having them reduced to the same denominator, and by reasoning over numerators, it is the same as if durations were *integer valued*. In addition, to simplify the study, we restrict ourselves to Timed Petri nets *without immediate transitions* (i.e. $\forall t\in\mathrm{T}, d(t)>0$), which is not so restrictive in real world practice and corresponds well to scheduling problems we are concerned with.

The transition firing semantics in TPN forbids *reentrance*. In other words, it is not possible to fire again a transition that has not yet *finished* to be fired. Again, this semantics is well fitted to scheduling problems, where transitions are associated to operations on machines. Thus, one can associate a unique *residual duration* to each transition without any possible confusion between several concurrent transitions activations. The residual duration vector is associated to the marking of a TPN to define its full state.

Definition 8 (TPN State). *Let* (R,d) *be a TPN. Its state* $e=(\vec{E_m},\vec{E_r})$ *is given by:*

- Its classical marking vector $\overrightarrow{E_m} \in \mathrm{N}^M$, associating to each place its number of tokens;
- A residual durations vector $\overrightarrow{E_r} \in \mathrm{N}^N$, associating to each *active* transition its remaining duration, and zero if the transition is not active.

The set of all states of a TPN is denoted by $\mathrm{S}(R,d)$. The fundamental concept that governs Timed Petri net behavior is the *controlled execution*, which associates to each transition the sequence of its successive firing dates.

Definition 9 (CE – Controlled Execution) *Let* (R,d) *be a TPN and* $t \in \mathrm{T}$ *a transition. A firing sequence for the timed transition* t*:* $(u_k^t) = u_1^t,\ldots,u_{K_t}^t \in \mathrm{N}$ *is an increasing sequence of firing dates, such that:*

$$\forall k \in [\![1, K_t - 1]\!], u_k^t + d(t) \le u_{k+1}^t \qquad (4)$$

A controlled execution is a family $(u_k^t)_{t\in\mathrm{T},k\in[\![1,K_t]\!]}$ *of firing sequences for all transitions of the TPN.*

Note that in the previous definition, equation (4) is used to forbid *reentrance*. For any transition t, k_t and $u_{K_t}^t$ may be infinite. Hereafter, we only consider finite CEs. We denote by $v_{\max}$ the ending date of the last firing in the CE: $v_{\max} = \max_{t\in\mathrm{T}}\left(u_{K_t}^t + d(t)\right)$. After $v_{\max}$, the state of the TPN under the considered CE will never change and we have: $\overrightarrow{E_r(v_{\max})} = \overrightarrow{0_N}$.

The formal expression of a CE is used to define several *characteristic vectors* allowing to verify the feasability of a CE. We assume that no transition is active at the initial state to simplify the formulation.

Definition 10 (Characteristic Vectors of Controlled Executions) *Let (R, d) be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at initial date* 0 *and* $(u_k^t)_{t\in\mathrm{T},k\in[\![1,K_t]\!]}$ *a controlled execution. Let* $v \in [\![0, v_{\max}]\!]$*. We define three characteristic vectors associated to* (u_k^t) *in the following way:*

$\overrightarrow{N(v)} \in \mathrm{N}^N$ is the vector corresponding to the number of firings that started within the interval $[0,v]$, defined by $\overrightarrow{N(v)}\Big|_t = card\left(\left\{u_{k,k\in[\![1,K_t]\!]}^t \mid u_k^t \le v\right\}\right)$;

- $\overrightarrow{D(v)} \in \mathrm{N}^N$ is the vector corresponding to the number of firings that started within the interval $[0,v[$, defined by $\overrightarrow{D(v)}\Big|_t = card\left(\left\{u_{k,k\in[\![1,K_t]\!]}^t \mid u_k^t < v\right\}\right)$;

- $\overrightarrow{F(v)} \in \mathrm{N}^N$ is the vector corresponding to the number of firings that ended within the interval $[0, v]$, defined by $\overrightarrow{F(v)}\Big|_t = card\left(\left\{u^t_{k,\, k\in[\![1,K_t]\!]} \mid u^t_k + d(t) \le v\right\}\right)$.

We have introduced above the definitions of *state* and *controlled execution* of a TPN. We define below how the state of a TPN is modified under a CE.

Definition 11 (Instantaneous State of a TPN under a Controlled Execution) *Let* (R, d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0 *and* $(u^t_k)_{t\in \mathrm{T}, k\in[\![1,K_t]\!]}$ *a controlled execution. Let* $v \in [\![0, v_{\max}]\!]$. *The instantaneous state* $e_v = \left(\overrightarrow{E_m(v)}, \overrightarrow{E_r(v)}\right)$ *at date* v *is given by:*

$$\overrightarrow{E_m(v)} = \overrightarrow{E_{m_0}} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{N(v)} \tag{5}$$

$$\forall t \in \mathrm{T}, \overrightarrow{E_r(v)}\Big|_t = \begin{cases} u^t_k + d(t) - v & \text{if } \exists k \in [\![1, K_t]\!] \; s.t. \; v \in [\![u^t_k, u^t_k + d(t)[\![\\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Informally, in the previous definition, the quantity $C^+ \cdot \overrightarrow{F(v)}$ corresponds to the tokens produced by the firings of transitions that ended strictly before the date v. Those tokens can be used to fire transitions *at date* v. The quantity $C^- \cdot \overrightarrow{N(v)}$ corresponds to the tokens used by the firings of transitions that started *until the date* v. Thus, the quantity $\overrightarrow{E_m(v)}$ corresponds exactly to the tokens remaining in the TPN at date v. The residual durations vector $\overrightarrow{E_r(v)}$ denotes the exact remaining time of transitions that are active at date v. Obviously, there can only be one $k \in [\![1, K_t]\!] \; s.t. \; v \in [\![u^t_k, u^t_k + d(t)[\![$ from equation (4). Note that (Chretienne, 1984) defines also the quantity $\overrightarrow{m^-(v)} = \overrightarrow{E_m(0)} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{D(v)}$. This quantity does not consider the tokens used by the firings of transitions that occur *exactly at date* v. Thus, it can be used to formulate the fireability condition for a transition in a TPN, independently from possible concurrent activations: under a controlled execution, a transition is fireable at date v iff $\overrightarrow{m^-(v)} \ge C^- \cdot \overrightarrow{e_t}$.

Obviously, like for Place/Transitions PNs, even if each transition is independently fireable at every date, the full CE is not necessarily valid as a whole since token may be used by several transitions at the same time. Thus, an improved condition for a CE to be feasible is given below.

Definition 12 (Feasible Controlled Execution). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0 *and* $(u_k^t)_{t\in \mathrm{T},\, k\in[\![1,K_t]\!]}$ *a controlled execution. This controlled execution is said to be feasible iff:*

$$\forall v \in [\![0, v_{\max}]\!], \overrightarrow{E_m(v)} \geq \overrightarrow{0_M} \tag{7}$$

The previous condition means that there must be enough tokens so that transitions may fire simultaneously.

2.2.3 Timed Petri Net Reachability Problem

Using the previous notations, the Timed Petri nets reachability problem consists in searching for a feasible CE allowing to reach a given final state from the initial state.

Definition 13 (Timed PN Reachability Problem). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0. *Let* $e_f = \left(\overrightarrow{E_{m_f}}, \overrightarrow{0_N}\right)$ *be a target state. The reachability problem for Timed Petri nets consists in finding a CE* $(u_k^t)_{t\in \mathrm{T},\, k\in[\![1,K_t]\!]}$ *such that* $e_{v_{\max}} = \left(\overrightarrow{E_m(v_{\max})}, \overrightarrow{E_r(v_{\max})}\right) = e_f$.

As said before, it is quite simple to see a parallelism between a scheduling problem and a Timed Petri net reachability problem. Indeed, let's consider for instance the Timed Petri net of Fig. 4. One remarks obviously that solving a reachability problem between markings $m_0 = \{p_1, p_2, p_3, m_1, m_2, m_3\}$ and $m_f = \{p_4, p_5, p_6, m_1, m_2, m_3\}$ means exactly finding a schedule of the production presented in the table on the left side.

Production To Schedule:

job 1:
$(m_1,2)(m_2,3)(m_3,4)$

job 2:
$(m_2,3)(m_3,2)(m_1,3)(m_3,3)$

job 3:
$(m_1,2)(m_3,4)(m_1,2)$

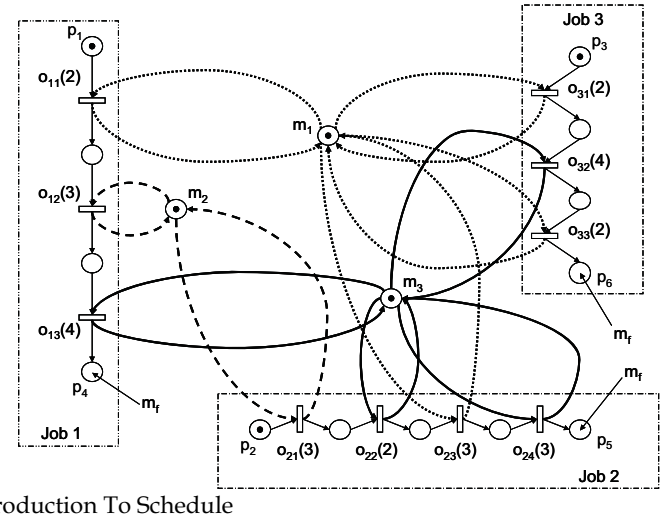


Fig. 4. TPN modelling a Production To Schedule

Several approaches have been proposed to solve the Timed Petri net reachability problem, either by restricting their study to a subclass of TPN, like Timed Event Graphs (where a place has exactly one input and one output transition), either by using dedicated heuristics. A complete bibliography can be found in (Richard, 2000).
Since the fire of a Timed transition can occur as soon as it is fireable and as late as one wants, there may exist, from a given state, an infinite number of reachable markings (depending on the time), and no reachability graph can be built. A first approach needs to consider Timed PN as a subclass of Time PN, in order to use the state enumeration methods (*state class graphs*) proposed by (Berthomieu and Diaz, 1991). On the other hand, when dealing with *early semantics* (a transition is fired as soon as it is fireable), it is possible to proceed to an enumerative and structural analysis (David and Alla, 1992).
The early semantics has been extensively studied for the special class of Timed Event Graphs, using (max,+) algebra (Bacceli et al., 1992). Since their structure does not handle conflicts, it is possible to obtain linear equations corresponding to the complete behaviour of the net.
In the following, we will show that our incremental approach can lead to mathematical programming models in the *most general* case.

3. Incremental approaches

As said before, the state equation (3) does not bring enough information to solve the reachability problem in all cases. This comes from the fact that it does not take into account the fireability conditions (1) of the individual transitions fired in the sequence σ. Incremental approaches improve this formulation by considering a given number of *step firings* corresponding to *parallel* and *reentrant* transitions. In this section, we discuss the interest of using steps and a fixed depth formulation.

3.1 Step based reachability formulation

Definition 14 (Step). *Let* R *be a Petri net. A step (Janicky and Koutny, 1991) is a multiset over the set of transitions* T. *We denote by* T^* *the set of steps built over* T.
Informally, a step is a set that can contain several copies of the same element, e.g. $\{t_1, t_1, t_2\}$, which we would note hereafter simply $2 \cdot t_1 + t_2$. We associate a step $\varphi = \sum_{j=1}^{N} \alpha_j \cdot t_j$ and its Parikh vector $\overrightarrow{\varphi}$ in the classical way, as a linear combination with non negative integer coefficients α_j of the Parikh vectors of each transition, i.e. $\overrightarrow{\varphi} = \sum_{j=1}^{N} \alpha_j \cdot \overrightarrow{e_{t_j}}$. A step is said *empty*, when $\varphi = \varnothing$, i.e. when $\forall j \in [1, N], \alpha_j = 0$.
Note that a step can contain the same transition more than once, corresponding to *transition reentrance*. Thus, when working with Timed Petri nets, steps would only mean that several *different* transitions are considered to be fired at the same time.
For a step to be fireable, its preceding marking must contain enough tokens so that each transition of the step may consume its *own* tokens, as described in the following definition.

Definition 15 (Step Firings). *Let* R *be a Petri net,* m *be a marking and* φ *be a step . The step* φ *is fireable from* m *iff:*

$$\overrightarrow{m} \geq C^{-}.\overrightarrow{\varphi} \tag{8}$$

If this condition is satisfied, the new marking m' reached from m by the firing of φ is defined as:

$$\overrightarrow{m'} = \overrightarrow{m} + C.\overrightarrow{\varphi} \tag{9}$$

Hereafter, we will use the notations already used previously: $m[\varphi\rangle$, $m_0[\varphi\rangle m_1$, $m_0[\varphi_1\varphi_2\ldots\varphi_k\rangle$ and $m_0[\varphi_1\varphi_2\ldots\varphi_k\rangle m_k$ to denote that a step or a step sequence is fireable, and the marking obtained in each case. The number of steps of a step sequence $\Phi = \varphi_1\varphi_2\ldots\varphi_K$ is denoted by $|\Phi| = K$.

The definition of step firings corresponds naturally to the firing of the underlying transitions. We will show that its use can lead to a formulation that is still equivalent to the initial PN behavior, but that can be more conveniently used in a mathematical programming framework. The following proposition explains the relation between step and transition firings with respect to reachability issues.

Proposition 2 (Step Reachability Equivalence). *Let* (R, m_0) *be a Petri net and* m_f *a marking.*

m_f is reachable from $m_0 \Leftrightarrow$

$$\begin{array}{l} \exists k \in \mathrm{N}, \\ \exists m_1, m_2, \ldots, m_{k-1} \in \mathrm{N}^M, \ s.t.: \\ \exists \varphi_1, \varphi_2, \ldots, \varphi_K \in \mathrm{T}^* \end{array} \begin{cases} \forall k \in [\![1, K-1]\!], & m_{k-1}[\varphi_k\rangle m_k \\ \wedge & m_{k-1}[\varphi_K\rangle m_f \end{cases} \tag{10}$$

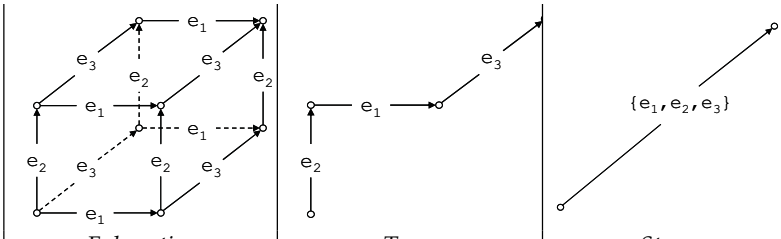
Proof. The proof of this proposition is not difficult but quite lengthly and hence is not given in this chapter. It can be found in a technical report available at url: http://www.eclille.fr/tomnab/asr07/. W

One must remark that the proof of the proposition 2 shows how it is possible to *construct* a firing sequence leading to m_f from a corresponding step sequence. Thus, to compute a firing sequence leading to a target marking, it will be sufficient to compute a step sequence leading to the same marking.

The main interest of our formulation is to *capture the parallelism* caused by the *interleaving of actions*, which is precisely one of the main advantages of using a Petri net as a model of a system. This issue has already been followed by (Vernadat et al., 1996), from whom we borrow the illustrative example of Fig. 5: ``*When two (*n *in the general case) components offer independent actions in parallel, the interleaving semantics expresses this behaviour by a diamond (an hypercube in the general case) within which each path lead to the same final state. Since all paths*

converge to the same state, the key idea is to develop only one particular path among the set of possible equivalent ones''.

Example 3 (Vernadat's steps). *As shown in Fig. 5, there exists several ways to handle 3 independent transitions from the point of view of the reachability graph. One can consider them one by one, which leads to handle 8 states and 12 firings. If we just refer to their corresponding Mazurkiewicz's trace (see (Vernadat et al., 1996) for details), we only have to handle 3 transition firings and 4 states. In the last case, one can capture the whole behavior in one unique firing that is called a step by Vernadat (with the meaning of ``footstep'').*



Point of view	*Exhaustive*	*Trace*	*Step*
# States	2^n	$(n+1)$	2
# Transition Firings	$n \times 2^n - 1$	n	1

Fig. 5. Some ways to handle independent transitions, from Vernadat

The characterization given in proposition 2 can be used to build a *mathematical programming model* based on steps which can be used to solve reachability-based PN analysis problems: one has just to express the right side of equation (10) using the linear equations (8) and (9) over integer variables. Such a model will be presented in section 4.

The advantage of using steps is that they allow to *reduce* the number of firings in our model – and then the number of variables – while keeping an equivalence with the initial properties. Thus it is not a *modification of the semantics* of PNs, but only a way to *capture the independence of transitions*. Of course, this reduction does not systematically holds, since it is easy to construct a Petri net where only one transition can be fired at a time. Thus, in the worst of cases, the step firings formulation may not bring any improvement as far as the *number* of firings used are concerned. However, this is a quite uncommon situation since it means that the Petri net does not show any *parallelism*.

3.2 Incremental search

We have seen the interest of using steps to formulate the reachability problem in PNs as a search for instanciations of integer variables constrained by a system of linear equations. This formulation allows us to use the paradigm of *mathematical programming* to solve the reachability problem. However, the *initial definition* of the reachability problem is not well adapted to the kind of formulation we propose to use, since definition 5 does not make any assumption concerning the number of steps needed to solve the reachability problem. In this paragraph, we define two *sub-problems* associated with the original reachability problem introduced before, which can be conveniently solved using the characterization of proposition 2 in a mathematical programming framework.

Definition 16 (Fixed Depth Reachability Problem). *Let* (R, m_0) *be a Petri net,* $k \in \mathrm{N}$ *and* m_f *a marking.*

$\mathrm{P}_1(k)$ *Find a step sequence allowing to reach the marking* m_f *from the marking* m_0 *in at most* k *steps .*

Definition 17 (Shortest Length Reachability Problem). *Let* (R, m_0) *be a Petri net, and* m_f *a marking reachable from* m_0 *.*

P_2 *Find the minimal length, denoted by* $K_{\min}$ *, of a sequence of steps allowing to reach the marking* m_f *from the marking* m_0 *.*

Of course, each of these sub-problems is *directly linked* to the initial one defined before, and each allows to solve a different kind of PN reachability analysis. For instance, the first formulation $\mathrm{P}_1(k)$ is highly useful for *model-checking* since it can serve to define an exhaustive search of the reachability graph. On the other hand, the second formulation P_2 is well designed to deal with *performance analysis* since it returns a firing sequence that *maximizes the parallelism* of the system. It can also give an helpful bound for the definition of *additional heuristics*. Finally, since it is clear that the complexity of the problem grows (w.r.t. number of variables and constraints) as the length k of the sequence of steps used increases, it seems also quite reasonable to search for the smallest value of the parameter k from which a solution exists.

The *fixed depth reachability problem* $\mathrm{P}_1(k)$ has already been studied by (Benasser, 2000) using the *logical abstraction technique*. His approach is based on the same notion of steps , but it uses constraint programming techniques. His algorithm iterates the number of steps used, adding one new step at each iteration, in order to test *all the lengths* of sequences of steps lower than k . Benasser proved that his algorithm is *correct* since the sequences found are effectively sequences of steps which produce the desired final marking. It is also *complete* since it can enumerate *all* the solutions of length smaller than a given integer k . In each iteration, the algorithm uses a mechanism of linear constraints solving. It has been implemented using the constraint logic programming software Prolog IV. The interest of using a constraint logic programming framework is that its resolution mechanism is *incremental* (Jaffar et al., 1992). Indeed, it is not necessary to redefine in each iteration the constraints incorporated into the previous stage. The constraints are added in the constraints solver so that it can reuse the results of the previous constraints propagation. The search for the concrete results is made at the end by an *enumeration* of all the possible integer solutions, which corresponds exactly to the sub-problem formulation $\mathrm{P}_1(k)$.

In section 4, we will adapt Benasser's algorithm to our own mathematical programming framework. To achieve the same kind of results, we will prove the *correctness* and *completeness* of our mathematical programming formulation with respect to $\mathrm{P}_1(k)$. These results will allow us to use integer linear programming techniques to find every solution of

$P_1(k)$. Some objective functions would also be defined to guide the search directly to an *optimal solution* in some way. Since P_2 can be easily expressed by iterating $P_1(k)$ instances for growing values of the parameter k, it will also be solved using the same technique.
Here, *Operational Research* techniques replace *Artificial Intelligence* ones, but the algorithm structure is the same. All techniques based on incremental approaches may share the same search algorithms. The most basic algorithm consists in searching in an incremental way amongst sequences the length of which are increased one by one.

3.2.1 Naive algorithm

This algorithm is fed with a *bound* K_{max} on what we call the ``*search depth*'' in order to prevent an *infinite loop*. Once chosen this value, the procedure generates iteratively a sequence of mathematical models of increasing size, and search for solutions in the corresponding search spaces using mathematical programming techniques. If there is no solution in less than k_{max} steps, the algorithm stops. It is described in Fig. 6.

1: $k \leftarrow 0$
2: **DO**
3: $k \leftarrow k+1$
4: Generate $\mathrm{MP}(k)$, a mathematical programming model for the problem $P_1(k)$ (which corresponds to characterization of proposition 2 with k steps).
5: Solve the model $\mathrm{MP}(k)$ using branch & bound techniques (e.g. Cplex solver). Let $\overrightarrow{X_{i[\![1,k]\!]}}$ be an optimal solution of $\mathrm{MP}(k)$ if it exists.
6: **IF** ($\mathrm{MP}(k)$ *has a solution*), **RETURN** $\overrightarrow{X_{i[\![1,k]\!]}}$
7: **WHILE** ($\mathrm{MP}(k)$ *is infeasible*) **AND** ($k \leq K_{\max}$)

Fig. 6. Naive Search Algorithm

During the formulation of the mathematical programming model at step 4, one should take care of the *domain of variables* representing the steps . Indeed: the definitions of step and step firings do not forbid *empty steps* leaving the markings unchanged. By considering empty steps valid in our formulations, we get the following result.

Proposition 3 (Satisfaction Monotony) *Let* $k \in \mathrm{N}$. *If the problem* $P_1(k)$ *is feasible, then for any integer* $k' \geq k$, *the problem* $P_1(k')$ *is also feasible.*

Proof. It is easy to construct a feasible solution for $P_1(k')$ from a feasible solution of $P_1(k)$ for $k' \geq k$ by adding empty steps. W
Note the same result would be true when dealing with the family of mathematical programming models $\mathrm{MP}(k)$: if there exists $k \in \mathrm{N}$ such that $\mathrm{MP}(k)$ admits a solution, any model $\mathrm{MP}(k')$ with $k' \geq k$ would be feasible too. This property motivates the jump search techniques proposed in the next paragraph.

3.2.2 Jump search

From proposition 3 and the definition of parameter $K_{\min}$, we get:

$$\begin{cases} \forall k < K_{\min} & \mathrm{P}_1(k) \text{ is infeasible} \\ \forall k \geq K_{\min} & \mathrm{P}_1(k) \text{ is feasible} \end{cases} \tag{11}$$

This property can help us to define new iterative techniques, since – for example – it shows that it is not necessary to solve all the problems $\mathrm{P}_1(k)$ for $k \leq K_{\min}$, like in the naïve search described before.

Of course, as said before, we must keep using an *incremental procedure* in order to avoid the use of large models if they are not needed. We propose finally some techniques based on *jumps* over the values of the search depth. These techniques allow to *decrease* the number of iterations needed, thus improving the search efficiency. Several jump strategies are possible. We describe briefly some elementary ones.

- **Forward jump search** The first family *continously increases* the value of the search depth. We can distinguish two main politics, depending on how the amplitude of jumps is defined.
  - **Fixed amplitude** Its value must be chosen in order to obtain a high exploration speed while minimizing the possible redundant steps. This type of strategy allows to estimate the profit precisely.
  - **Dynamic amplitude** This second strategy uses *variable amplitudes*. Increasing amplitudes should be used for small values of k, and decreasing ones when the exploration becomes more difficult. This kind of behavior is less easily quantifiable.

  These politics both can lead to overtake $K_{\min}$ when a solution is found. In this case, it is not anymore possible to answer precisely the problem P_2, since we do not get the exact value of $K_{\min}$. To compensate this lack of information, one can use a dichotomic search.
- **Dichotomic search** This kind of procedure needs to know a maximal bound for k. Its value is given by a previous successful execution of the forward jump search.

The main interest of jump search is that it allows to win in *efficiency*. Since we do not know the number of steps needed to find a solution if it exists, the use of such a technique allows us, when it is possible, not to have to develop the entire set of formulations of length lower than $K_{\min}$. Numerical experiments show that even if the *size* of models is increasing, the corresponding *practical complexity* does not always follows the same evolution.

Finally, it must be said that the procedure described in Fig. 6 is only a *semi-complete* one. Indeed, in the context of *unbounded* PNs, the value of $k_{\max}$ is set arbitrarily, as we do not know any information on the number of steps needed to find a possible solution. Thus, if *no solution* is obtained *before* the value of k has been reached, one *cannot conclude* on the reachability property.

To the contrary, when dealing with bounded PNs, it is possible to set $K_{\max}$ to the value of the *sequential depth* of the net, a parameter we have defined in (Bourdeaud'huy et al., 2004a) and which *guarantee* the *complete exploration* of the reachability graph. Using this parameter as search depth, it is *always possible* to conclude when the algorithm stops.

3.3 Adaptation to timed Petri nets

We have seen in the previous section the awaited benefits from using an incremental approach made of step firings. Before introducing the mathematical models in section 4, we propose to adapt the step based formulation to Timed Petri nets.

We start by adapting the previous formalism to Timed Petri nets. The key idea is again to consider the evolution of a Timed Petri net `` *step by step* ''.

Definition 18 (Timed step). *Let* (R,d) *be a Timed Petri net. A timed step is a pair* $\psi=(\varphi,v)$ *such that:*

- $\varphi=\sum_{j\in[\![1,n]\!]}\alpha_j\cdot t_j$ *is a step* $\in \mathrm{T}^*$ *for the Place/Transition Petri net* R*, such that* $\forall j\in[\![1,N]\!],\alpha_j\in\{0,1\}$*;*
- v is a date $\in \mathrm{N}$.

The set of all timed steps of a Timed Petri Net is denoted by $\mathrm{T}^*_{\mathrm{TPN}}$.

Definition 19 (Timed steps Firings). *Let* (R,d) *be a Timed Petri net. Let* $e=(\overrightarrow{E_m},\overrightarrow{E_r})$ *be a state given at date* v*. Let* $v'\geq v$ *and* $\Delta_v=v'-v\in\mathrm{N}$*. The timed step* $\psi=(\varphi,v')$ *is fireable from e iff:*

$$\forall t\in\varphi,\overrightarrow{E_r}\Big|_t\leq\Delta_v \tag{12}$$

$$C^-\cdot\overrightarrow{\varphi}\leq\overrightarrow{E_m}+C^+\cdot\sum_{\substack{t\in\mathrm{T},\\ 0<\overrightarrow{E_r}\big|_t\leq\Delta_v}}\overrightarrow{e_t} \tag{13}$$

If this condition is satisfied, the new state $e'=(\overrightarrow{E'_m},\overrightarrow{E'_r})$ *reached at date* v' *from* e *by the firing of* $\psi=(\varphi,v')$ *is defined as:*

$$\overrightarrow{E'_m}=\overrightarrow{E_m}-C^-\cdot\overrightarrow{\varphi}+C^+\cdot\sum_{\substack{t\in\mathrm{T},\\ 0<\overrightarrow{E_r}\big|_t\leq\Delta_v}}\overrightarrow{e_t} \tag{14}$$

$$\forall t\in\mathrm{T},\overrightarrow{E'_r}\Big|_t=\begin{cases}d(t) & \text{if } t\in\varphi\\ \overrightarrow{E_r}\big|_t-\Delta_v & \text{if } \overrightarrow{E_r}\big|_t-\Delta_v>0\\ 0 & \textit{otherwise}\end{cases} \tag{15}$$

The above definition follows the firing semantics of Timed Petri nets described above, from the point of view of a punctual firing between two markings. Informally, equation (12) means that a transition fired within a step must not be active at the time of the firing, in order to comply with the *non-reentrance* hypothesis. Equation (13) verifies that there is enough tokens *at date v'* to fire the step φ. The set $\mathrm{A} = \{t \in \mathrm{T}, 0 < \overrightarrow{E_r}\big|_t \leq \Delta_v\}$ used in equations (13) and (14) denotes the transitions that were active at date v and are no longer alike at date v'. Thus, the quantity $\sum_{\mathrm{A}} C^+ \cdot \overrightarrow{e_t}$ corresponds to the tokens that would appear between dates v and v'. At last, the update of the residual durations vector at equation (15) is made as follows:

- If the transition t is fired in the step φ, its duration is used to initialize the corresponding component of the residual vector;
- If a transition that was previously active is *still active* at date v', its residual duration is updated by taking into account the time elapsed from the previous date;
- Otherwise, if a transition was active and has finished, or if a transition was not active and is not fired in the step φ, its residual duration is null.

As above, we will use the notations $e[\psi\rangle$, $e_0[\psi\rangle e_1$, $e_0[\psi_1\psi_2\ldots\psi_k\rangle$ and $e_0[\psi_1\psi_2\ldots\psi_k\rangle e_k$ to indicate that a timed step or a timed step sequence is fireable, and the state obtained in each case. We give finally below the main proposition concerning the use of timed steps in the context of Timed Petri nets.

Proposition 4 (Equivalence between Controlled Executions and timed step firings). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* $v_0 = 0$. *Let* $e_f = \left(\overrightarrow{E_{m_f}}, \overrightarrow{0_N}\right)$ *be a state.*

There exists a feasible controlled execution allowing to reach e_f *at date* $v_{\max}$ *from* $e_0 \Leftrightarrow$

$$\begin{array}{l} \exists k \in \mathrm{N}, \\ \exists v_1, v_2, \ldots, v_K \in \mathrm{N} \\ \exists e_1, e_2, \ldots, e_K \in \mathrm{S}(R,d). \\ \exists \psi_1 = (\varphi_1, v_1), \psi_2, \ldots, \psi_K = (\varphi_K, v_K), \\ \qquad \psi_{\max} = (\overrightarrow{0_N}, v_{\max}) \in \mathrm{T}^*_{\mathrm{TPN}} \end{array} \quad s.t.: \begin{cases} \forall k \in [1, K], e_{k-1}[\psi_k\rangle e_k \\ e_K[\psi_{\max}\rangle e_f \end{cases} \tag{16}$$

Proof. Here again, the reader is referred to the technical report available at http://www.eclille.fr/tomnab/asr07/ for the complete proof. W

Following the previous proposition, it is sufficient to search for timed step sequences to solve the reachability problem in TPN. The advantage of such an approach is obvious: like for basic PN, it is well adapted to the definition of a mathematical programming model with a reduced number of variables and constraints, since it allows to consider explicitly *parallel*

executions. Moreover, it is *incremental* since one can progressively increase the number of steps used in the formulation without redefining the whole set of constraints.
Since reachability by controlled execution or timed step sequence is equivalent, we define as above a *sub-problem* of the TPN reachability problem, which can be conveniently solved using the characterization of proposition 4 in a mathematical programming framework.

Definition 20 (Fixed Depth Timed PN Reachability Problem). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0 *. Let* $e_f = \left(\overrightarrow{E_{m_f}}, \overrightarrow{0_N}\right)$ *be a target state.*

$\mathrm{TP}_1(v)$ *Find a timed step sequence allowing to reach the state* e_f *from the state* e_0 *in at most* v *timed steps.*

In the next section, we prove the correctness and completeness of our mathematical programming model with respect to this formulation.

4. Integer linear programming models

In this section, we give integer linear programming models corresponding to the characterizations introduced in propositions 2 and 4.

4.1 Place/transition Petri nets

4.1.1 Integer linear programming model

In the previous section, we have shown that step sequences are sufficient to prove that a marking is reachable and to find the firing sequence leading to it. In this section, we show how the search for a step sequence can be expressed as *a mathematical programming problem*. We prove also that this model is *correct and complete* with respect to the fixed depth reachability problem $\mathrm{P}_1(k)$.

Our integer programming model is directly built from equations and inequalities (8), (9) and (10). Thus we get:

Model 1 (Integer Programming Model). *Let* (R,m_0) *be a Petri net with* $\mathrm{P} = \{p_1,\ldots,p_M\}$, $\mathrm{T} = \{t_1,\ldots,t_N\}$, m_f *a marking and* $K \in \mathrm{N}$ *. The integer linear program* $\mathrm{IP}(K)$ *is defined by:*

$\mathrm{IP}(K)$

$$\text{Minimize } \omega(\overrightarrow{X}) \tag{17a}$$

subject to:

$$\forall i \in [\![1,K]\!], \sum_{j=1}^{i-1} C \cdot \overrightarrow{X_j} - C^- \cdot \overrightarrow{X_i} \geq -\overrightarrow{M_0} \tag{17b}$$

$$\sum_{i=1}^{K} C \cdot \overrightarrow{X_i} = \overrightarrow{M_f} - \overrightarrow{M_0} \tag{17c}$$

$$\forall i \in [\![1,K]\!], \quad \forall j \in [\![1,N]\!], \quad X_{ij} \in \mathrm{N} \tag{17d}$$

In the model $\mathrm{IP}(K)$, variables X_{ij} represent the components of K steps in the sense of definition 14. Inequalities (17b) correspond to the combination of fireability (8) and

reachability (9) conditions presented in definition 15. To reduce the size of the problem, the variables $(m_i)_{\square 1,K\square}$ of equation (10) have been dropped by using the substitution of equation (9). Equation (17c) corresponds to the target marking to reach like in (10). The integrality constraints of variables X_{ij} are expressed by (17d). The expression of objective function $\omega(\overrightarrow{X})$ will be described later.

The following proposition is *central* in our construction: it shows that our model $\mathrm{IP}(k)$ characterizes *all* the solutions of the problem $\mathrm{P}_1(k)$.

Proposition 5 (Correctness and completeness of *IP(k)* w.r.t. *$P_1(k)$*). *Let (R, m_0) be a PN and $k \in \mathrm{N}$. Then we have:*

- *Any solution of $\mathrm{IP}(k)$ is also a solution of $\mathrm{P}_1(k)$ (Correctness)*
- *Any solution of $\mathrm{P}_1(k)$ can be expressed as a solution of $\mathrm{IP}(k)$ (Completeness)*

Proof. Those results come directly from the proof of proposition 2. W

According to proposition 5, to solve the fixed-depth PN reachability problem is equivalent to search for the solutions of an integer linear programming problem. In this way, the exploration of the reachability graph and the resolution of the corresponding reachability problems are reduced to the resolution of a system of equations. The ``*combinatorially explosive*'' reachability graph is reduced to a sequence of ``*abstract*'' steps . Since $\mathrm{IP}(k)$ needs $k.n$ variables and $(k+1)\cdot M$ constraints, the size of the $\mathrm{IP}(k)$ model grows *linearly* with the value of the parameter k. This having been said, two remarks must be done.

- Of course, we do not pretend that the combinatorial explosion problem has been *wiped out*. It is only *postponed* to the mathematical programming solution phase. Meanwhile, our formulation allows to benefit automatically from today's best solvers.
- On the other hand, compared to many other exploration techniques, one of the interests of our formulation is to avoid the ``*exploration*'' of the branches of the graph which do not lead to the *desired final marking*. Indeed, these cases are ``*automatically cut off*'' by the equation (17c).

From these two statements, we have good reasons to think that our method will bring some improvements on the research topic considered. We have developed several *additional mechanisms* to validate our approach and to improve the performance of the solution. All these improvements have been formally defined in (Bourdeaud'huy et al., 2004a,b,c, 2007; Bourdeaud'huy, 2004). We describe them below and refer the reader to these papers for more information.

4.1.2 Practical improvements

Objective Functions: It is obvious that $\mathrm{P}_1(k)$ has a solution if and only if the feasible set of $\mathrm{IP}(k)$ problem is non empty. This property stays true for *any objective function*.

Nevertheless, the efficiency of solving $\mathrm{IP}(k)$ *may depend* on the objective function chosen. Let's remark that some objective functions are more usefull in practice than others.
For example, if there is no difference between solutions, a *constant function* is valid, which leads to the selection of the first feasible solution found. We define thus the function *vanishing identically*:

$$\forall \overrightarrow{X}, obj_1(\overrightarrow{X}) = 0$$

This objective function leads to the selection of the first feasible solution found, thus to the best practical performances. We could also use *physical sense* objective functions according to the particular context of the studied problem. For example, we could differentiate the solutions by their norm, considering thus the L_1 *norm of steps*:

$$\forall \overrightarrow{X}, obj_2(\overrightarrow{X}) = \sum_{i=1}^{K} \mathrm{P}\overrightarrow{X_i}\,\mathrm{P}_1 = \sum_{i=1}^{K}\sum_{j=1}^{N} X_{ij} \tag{18}$$

The function obj_2 allows to search for the ``*fastest*'' sequence, in terms of number of firings.

Relaxations: In order to decrease the time needed to conclude on the infeasibility of the $\mathrm{IP}(k)$ problem, we propose to use *relaxation techniques*. They are useful in the field of combinatorial optimization. The principle of these techniques is to replace the complex original problem by one or several simpler ones.

Definition 21 (Relaxation). *A relaxation of an optimization problem P of type maximisation is an optimization problem R such that:*

- *Each feasible solution for P is also a feasible one for R ;*
- *The value of the objective function of any solution of R is greater than or equal to the value of the objective function of the same solution for P .*

Among useful properties of the relaxation and duality techniques in solving an optimization problem P is that the optimal value of the relaxation problem provides an upper bound on the optimal value of the corresponding P. Moreover, if the relaxed problem is infeasible then the problem P is also infeasible. In our context, this second property is used to conclude that the $\mathrm{IP}(k)$ problem is infeasible by solving a relaxed problem before reaching $K_{\min}$.

- *LP-relaxation* consists in relaxing integrality constraints. Bounds derived from other relaxations can be stronger than those obtained from LP. In the literature (Parker and Rardin, 1988), Lagrangean relaxation, surrogate relaxation and composite relaxation are usually used to obtain such upper bounds.
- *Lagrangean relaxation* consists to relax complicating contraints and incorporating them into the objective function with a so-called Lagrangean multiplier (Geoffrion, 1974). However, note that relaxing fireability (17b) or reachability (17c) constraints is ``*too strong*'' from the physical point of view. Indeed: without fireability conditions, the modified model represents only the *state equation*, which is already supposed to have

solutions. In the other hand, dealing with a model without the reachability constraint correspond to study the *whole behaviour* of the net. Any sequence of fireable transitions of the correct length is solution to the relaxed problem and do not bring information.

- *Surrogate relaxation* replaces the original constraints by a single new one, called a surrogate constraint (Glover, 1977).

Binary Programming Model: We propose also a binary programming model denoted by $\mathrm{BIP}(K)$. In this model, equation (17d) is replaced by:

$$\forall i \in [\![1, K]\!], \forall j \in [\![1, N]\!], X_{ij} \in \{0,1\}$$

This formulation is still *correct* since it corresponds to a *restriction* of the initial model $\mathrm{IP}(K)$: parallel behaviours are still allowed, but reentrance is forbidden. This formulation may be *more efficient* since the domain of variables is reduced. However, it may be necessary to fire more steps to reach some markings than using the $\mathrm{IP}(K)$ model. Nevertheless, it is quite simple to show that this last model preserves an *equivalence* with the $\mathrm{IP}(K)$ one. Indeed, any solution of $\mathrm{BIP}(K)$ is also obviously a solution of $\mathrm{IP}(K)$. Inversely, any solution of $\mathrm{IP}(K)$ could be obtained by an aggregation of a solution of $\mathrm{BIP}(K')$, with $K' \geq K$ (some steps in a solution of $\mathrm{IP}(K)$ may have to be splitted into several firings of what could be called ``*binary steps*'' of the solution from $\mathrm{BIP}(K')$). Thus we are able to use the model $\mathrm{BIP}(K')$ in the same way as $\mathrm{IP}(K)$. The only difference is the value of $K'_{\min}$ associated to $\mathrm{BIP}(K')$ which can be greater than $K_{\min}$.

Empty Steps Management: As said before, to consider empty steps valid in our formulations bring interesting theoretical results. However, practically speaking, empty steps do not bring useful information considering the resolution of the reachability problem, since they do not change the markings. We propose thus several additional constraints dedicated to the management of empty steps .

- In the binary model, one can add an extra linear constraint in order to express a notion of *partial order* in the steps:

$$\forall i \in [\![1, K-1]\!], N \cdot \sum_{j=1}^{N} X_{ij} \geq \sum_{j=1}^{N} X_{(i+1)j} \tag{19}$$

These constraints mean that empty steps have to appear *at the end* of the constrained sequence. Indeed: if a step X_i is empty (i.e. $\forall j \in [\![1, N]\!], X_{ij} = 0$), all its successors $\left(X_k \ s.t. \ k \in [\![i+1, K]\!]\right)$ in the step sequence have to be empty in a ``*chain reaction*''. Inversely, since we consider binary steps, equation (19) is obviously true when $\exists j \in [\![1, N]\!]$ s.t. $X_{ij} = 1$, because then $N \cdot \sum_{j=1}^{N} X_{ij} \geq N$.

- We propose also a new objective function, which corresponds to maximize the *number of empty steps* in the feasible sequence. This is equivalent to minimize the number of *non-empty* steps obj_3 defined below:

$$\forall \overrightarrow{X}, obj_3(\overrightarrow{X}) = card\left\{ i \in [\![1, K]\!] \; s.t. \sum_{j=1}^{N} X_{ij} \neq 0 \right\} \tag{1}$$

The function obj_3 allows us to look for the *shortest solutions* in term of ``*equivalent length*'' when the empty steps have been removed. To model obj_3, we introduce new intermediate *binary* variables $\alpha_1, \alpha_2, \ldots, \alpha_k$ and we incorporate in the linear programming models the following additional constraints:

$$\forall i \in [\![1, K]\!], \quad 1 + (\alpha_i - 1) \cdot B \leq \sum_{j=1}^{N} X_{ij} \leq \alpha_i \cdot B$$

$$\forall i \in [\![1, K]\!], \qquad \alpha_i \in \{0,1\}$$

where B is a sufficiently big number, chosen much bigger than the number of transitions and tokens in the net, e.g. $B \geq M \cdot N \cdot K$. Since the variables $(\alpha_i)_{i \in [\![1,K]\!]}$ are binary, it is easy to check that $\alpha_i = 0$ iff the step X_i is empty (a complete proof is given below in proposition 6). Thus the objective function obj_3 can be expressed in the following way:

$$\forall X, obj_3(X) = \sum_{i=1}^{K} \alpha_i$$

- Finally, one could simply *forbid* empty steps by adding to the models the following constraint:

$$\forall i \in [\![1, K]\!], \mathrm{P}\overrightarrow{X_i}\,\mathrm{P}_1 \geq 1$$

Decomposition Technique: To improve the performance of the resolution, we have proposed constraints corresponding to a *decomposition technique* based on a *partition of the state space* according to the solutions of the PN *state equation*. For each solution $\overrightarrow{\sigma}$ of the underlying state equation – defined between the same initial and final markings –, we add the constraint:

$$\forall j \in [\![1, N]\!], \sum_{i=1}^{K} X_{ij} = \overrightarrow{\sigma}\Big|_{t_j}$$

Such a decomposition technique allows to adress the complexity of the problem in two steps:

- The first step uses well known *T-invariants computation techniques* (see for example (Colom and Silva, 1989a)) which are independent from the initial marking, and thus allows to reuse the same information for many different initial and final markings.
- Given the Parikh vector of the whole firing sequence to discover, the resolution of the reachability problem should be slighty simple. However, one should note that this second step remains difficult: it is not sufficient to distribute the firings over the steps since *each step* must be fireable. Moreover, developping heuristics methods is challenging since *deadlock situations* can occur late after a bad choice has been made. Finally, the mathematical programming approaches proposed here are well designed to handle the second step of the search.

4.1.3 Numerical experiments

Numerous practical experiments were led in (Bourdeaud'huy et al., 2004a,b,c, 2007; Bourdeaud'huy, 2004) in order to assess the efficiency of our mathematical programming models. There is no space left to copy them all here, but several results must be pointed out.

- We have compared the influence of the objective functions obj_1, obj_2 and obj_3. The corresponding results were quite foreseeable: obj_1 led to the best results, followed by obj_2 and finally obj_3. However, it must be said that the performance of resolution using obj_3 was very weak, even for the smallest instances of our families, since the size of the corresponding models is large. To the contrary, the performance of models using obj_2 were quite close to the basic performance given by obj_1, about 3 times worse in a pinch.
- Our experiments to validate the pertinence of the *LP-relaxation* shown that for the whole set of PN studied, the gap between $K_{\min}$ for the *LP-relaxation* and the integer model is small. Such statement suggests a property of *integrality* of the kind of problems considered – i.e. continuous solutions are somehow integer –, which may be interesting to study.
- Compared one with another, model BIP behaves better than IP. Of course, one could build an example for which the contrary would holds. Nevertheless, this observation were made over the whole set of our experiments.
- Dealing with decomposition technique, preliminary experiments have shown that a search inside a given equivalence class is 20% faster than for the whole state space.
- We have compared our technique with other classical tools from the PN community: Ina (Roch and Starke, 2002) and Netched (Benasser, 2000).
    - Ina means *Integrated Net Analyzer*. It is an analysis tool which allows the computation of firing sequences between markings thanks to the exploration of a covering graph. It implements some reduction techniques, e.g. persistent sets (Valmari, 1991) and symmetries (Schmidt, 1998).
    - Netsched is the implementation of the logical abstraction technique developped by Benasser. It has been implemented using the constraint logic programming

language Prolog IV.

Our approach has shown very good results and dominates the other tools on some instances. However, there exists some special instances – see for example (Bourdeaud'huy et al., 2004c) – for which our method is dominated by Netsched, particularly when the underlying reachability graph is sparse.

In the next section, we develop a similar mathematical programming approach for Timed Petri nets.

4.2 Timed Petri nets

We proceed as for Place/Transition PN, by adapating the characterization proposition 4 to build a mathematical programming model. For that, we need to *linearize* the equations defining timed step firings. We introduce in the next section two operators and the corresponding linearization variables and equations that we use to obtain the linear integer programming model.

4.2.1 Discrimination operators

We start by giving a useful proposition, which has been already used above dealing with the formulation of objective function obj_3.

Proposition 6 (Discrimination Variables). *Let* $X \in \mathrm{S} \subset \mathrm{Z}$ *and* $B \in \mathrm{N}^*$ *be ``sufficiently large''. Let* $\alpha \in \{0,1\}$ *and* $\beta \in \mathrm{N}$ *such that:*

$$1 + (\alpha - 1) \cdot B \le X \le \alpha \cdot B \tag{20a}$$

$$X \le \beta \tag{20b}$$

$$\beta \le \alpha \cdot B \tag{20c}$$

$$\beta \le B \cdot (1 - \alpha) + X \tag{20d}$$

Then we have:

$$\begin{cases} X > 0 & \Rightarrow \quad \alpha = 1 \text{ et } \beta = X \\ X \le 0 & \Rightarrow \quad \alpha = 0 \text{ et } \beta = 0 \end{cases}$$

Proof. Let's assume that X is strictly positive. The right side of inequation (20a) implies then $1 \le X \le \alpha \cdot B$, i.e. $\alpha \ge \frac{1}{B} > 0$. Thus we have $\alpha = 1$, and the left side of inequation (20a) is valid. The inequation (20d) implies then $\beta \le X$, and inequation (20b) implies $\beta = X$. Inequation (20c) is valid only if B is sufficiently large, namely: $B \ge \max_{X \in \mathrm{S}}(X)$.

Conversely, if X is negative or null, the left side of inequation (20a) implies $1 + (\alpha - 1) \cdot B \le 0$, which implies $\alpha \le 1 - \frac{1}{B}$ < 1. We have then $\alpha = 0$ and the right

side of inequation (20a) is valid. The inequation (20c) implies then $\beta \leq 0$, i.e. $\beta = 0$. Inequation (20b) is valid and inequation (20d) is valid if B is sufficiently large, again if $B \geq \max_{X \in \mathrm{S}}(|X|)$. W

Using 2 variables and 5 equations per unknown X, one can thus obtain *linearly* X^s and X^+, corresponding to the ``*sign*'' of $X$ and its ``*positive component*''. We *extend* these operators ``+'' and ``s'' to *vector* objects, by applying them uniformly on each component of the considered vector.

Definition 22 (Discrimination Operators). *Let* $k \in \mathrm{N}$ *and* $\vec{x} \in \mathrm{Z}^k$. *We denote by*:

- $\vec{x}^+ \in \mathrm{N}^k$ *the vector of its positive components, such that* $\forall c \in [\![1,k]\!], \vec{x}^+(c) = \vec{x}(c)$ *if* $\vec{x}(c) > 0$ *and 0 otherwise;*
- $\vec{x}^s \in \{0,1\}^k$ *the vector representing its sign, such that:* $\forall c \in [\![1,k]\!], \vec{x}^s(c) = 0$ *if* $\vec{x}(c) \leq 0$ *and* $\vec{x}^s(c) = 1$ *otherwise.*

Since the operators above are easily expressed using linear equations, we use them to *reformulate* the characterization proposition 4. The new formulation will be used to build a linear programming model corresponding to the firing of a timed step sequence.

4.2.2 Timed steps linear formulation

In this section, we consider the equations (12) to (15) defining timed step firings and reformulate each of them using the discrimination operators given above. In order to avoid confusions of notations, we use lower case letters to denote the state vectors $(\vec{e_m}, \vec{e_r})$ expressed using discrimination operators.

Proposition 7 (Timed step Firings Reformulation). *Let* (R,d) *be a Timed Petri net. Let* $e = (\vec{e_m}, \vec{e_r})$ *be a state given at date* v. *Let* $v' \geq v$ *and* $\Delta_v = v' - v \in \mathrm{N}$. *Let* $\psi' = (\varphi', v')$ *be a timed step . Then we have:*

$$e[\psi'\rangle \Leftrightarrow \begin{cases} \vec{\varphi'} \leq \vec{1_n} - (\vec{e_r} - \Delta_v \cdot \vec{1_n})^s & (21) \\ \vec{e_m} - C^- \cdot \vec{\varphi'} + C^+ \cdot (\vec{e_r}^s - (\vec{e_r} - \Delta_v \cdot \vec{1_n})^s) \geq \vec{0_m} & (22) \end{cases}$$

$$\begin{aligned} & e[\psi'\rangle e' \\ & e' = (\vec{e'_m}, \vec{e'_r}) \end{aligned} \Leftrightarrow e[\psi'\rangle \wedge \begin{cases} \vec{e'_m} = \vec{e_m} - C^- \cdot \vec{\varphi'} + C^+ \cdot (\vec{e_r}^s - (\vec{e_r} - \Delta_v \cdot \vec{1_n})^s) & (23) \\ \vec{e'_r} = (\vec{e_r} - \Delta_v \cdot \vec{1_n})^+ + \sum_{t \in \mathrm{T}} d(t) \cdot \vec{\varphi'}\Big|_t \cdot \vec{e_t} & (24) \end{cases}$$

Proof. To prove the above proposition, one has just to verify that equations (21) to (24) correspond exactly to equations (12) to (15) from definition 19. W

4.2.3 Illustrative example

		$v_0 = 0, \Delta_{v_0} = 1$			$v_1 = 1, \Delta_{v_1} = 3$			$v_2 = 4, \Delta_{v_2} = 2$		
places	$\overrightarrow{e_m}(0)$	$C^- \cdot \overrightarrow{\varphi_{v_0+\Delta_{v0}}}$	$C^+ \cdot (\overrightarrow{e_r}(v_0)^s - (\overrightarrow{e_r}(v_0) - \Delta_{v0} \cdot \overrightarrow{1_N})^s)$	$\overrightarrow{e_m}(v_0 + \Delta_{v0}) = \overrightarrow{e_m}(v_1)$	$C^- \cdot \overrightarrow{\varphi_{v_1+\Delta_{v1}}}$	$C^+ \cdot (\overrightarrow{e_r}(v_1)^s - (\overrightarrow{e_r}(v_1) - \Delta_{v1} \cdot \overrightarrow{1_N})^s)$	$\overrightarrow{e_m}(v_1 + \Delta_{v1}) = \overrightarrow{e_m}(v_2)$	$C^- \cdot \overrightarrow{\varphi_{v_2+\Delta_{v2}}}$	$C^+ \cdot (\overrightarrow{e_r}(v_2)^s - (\overrightarrow{e_r}(v_2) - \Delta_{v2} \cdot \overrightarrow{1_N})^s)$	$\overrightarrow{e_m}(v_2 + \Delta_{v2})$
p_1	1	1	0	0	0	0	0	0	0	0
p_2	0	0	0	0	1	1	0	0	0	0
p_3	0	0	0	0	0	0	0	0	0	0
p_4	0	0	0	0	0	0	0	0	0	0
p_5	1	0	0	1	1	0	0	0	0	0
p_6	0	0	0	0	0	0	0	0	1	1
r_1	1	1	0	0	1	1	0	0	1	1
r_2	1	0	0	1	1	0	0	0	0	0

Fig. 7. Intermediary Calculations (Markings)

			$v_0 = 0, \Delta_{v_0} = 1$						$v_1 = 1, \Delta_{v_1} = 3$						$v_2 = 4, \Delta_{v_2} = 2$					
t_4 t_3 t_2 t_1	transitions	$d(t)$	$\overrightarrow{\varphi_{v_0+\Delta_{v0}}}$	$\overrightarrow{e_r}(v_0)^s$	$\overrightarrow{e_r}(v_0) - \Delta_{v0} \cdot \overrightarrow{1_N}$	$(\overrightarrow{e_r}(v_0) - \Delta_{v0} \cdot \overrightarrow{1_N})^s$	$(\overrightarrow{e_r}(v_0) - \Delta_{v0} \cdot \overrightarrow{1_N})^+$	$\sum_{t \in \mathbb{T}} d(t) \cdot \overrightarrow{\varphi_{v_0+\Delta_{v0}}}(t) \cdot \overrightarrow{e_t}$	$\overrightarrow{e_r}(v_0)^s - (\overrightarrow{e_r}(v_0) - \Delta_{v0} \cdot \overrightarrow{1_N})^s$	$\overrightarrow{e_r}(v_0 + \Delta_{v0}) = \overrightarrow{e_r}(v_1)$	$\overrightarrow{\varphi_{v_1+\Delta_{v1}}}$	$\overrightarrow{e_r}(v_1)^s$	$\overrightarrow{e_r}(v_1) - \Delta_{v1} \cdot \overrightarrow{1_N}$	$(\overrightarrow{e_r}(v_1) - \Delta_{v1} \cdot \overrightarrow{1_N})^s$	$(\overrightarrow{e_r}(v_1) - \Delta_{v1} \cdot \overrightarrow{1_N})^+$	$\sum_{t \in \mathbb{T}} d(t) \cdot \overrightarrow{\varphi_{v_1+\Delta_{v1}}}(t) \cdot \overrightarrow{e_t}$	$\overrightarrow{e_r}(v_1)^s - (\overrightarrow{e_r}(v_1) - \Delta_{v1} \cdot \overrightarrow{1_N})^s$	$\overrightarrow{e_r}(v_1 + \Delta_{v1}) = \overrightarrow{e_r}(v_2)$	$\overrightarrow{\varphi_{v_2+\Delta_{v2}}}$	$\overrightarrow{e_r}(v_2)^s$	$\overrightarrow{e_r}(v_2) - \Delta_{v2} \cdot \overrightarrow{1_N}$	$(\overrightarrow{e_r}(v_2) - \Delta_{v2} \cdot \overrightarrow{1_N})^s$	$(\overrightarrow{e_r}(v_2) - \Delta_{v2} \cdot \overrightarrow{1_N})^+$	$\sum_{t \in \mathbb{T}} d(t) \cdot \overrightarrow{\varphi_{v_2+\Delta_{v2}}}(t) \cdot \overrightarrow{e_t}$	$\overrightarrow{e_r}(v_2)^s - (\overrightarrow{e_r}(v_2) - \Delta_{v2} \cdot \overrightarrow{1_N})^s$	$\overrightarrow{e_r}(v_2 + \Delta_{v2})$
2 5 4 3																				
0 0 0 0	$\overrightarrow{e_r}(0)$																			
0 0 0 1			0	0	-1	0	0	0	0	1	0	0	-3	0	0	0	0	0	0	0
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	0	0	-2	2	-2
-1 -1 -1 -1			-1	0	-2	0	0	-3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 3			0	0	-1	0	0	3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 3			0	0	-1	0	0	3	3	0	0	0	-3	0	0	1	0	-2	2	-2
1 0 1 0			1	0	0	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 1			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
-3 -3 -3 0			-3	0	-4	0	0	-3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
2 0 4 0			2	0	0	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 1			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
2 0 4 0			2	0	0	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
1 0 1 0			1	0	0	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 -2 2 -2			0	-2	-3	0	0	-3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 1 0			0	0	-3	0	0	-3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 2 0			0	0	-4	0	0	-3	3	0	0	0	-3	0	0	1	0	-2	2	-2
0 0 0 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
1 0 0 0			1	0	0	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2
0 0 2 0			0	0	-1	0	0	0	0	1	0	0	-3	0	0	1	0	-2	2	-2

Fig. 8. Intermediary Calculations (Residual Durations)

In order to help understanding the above equations, we propose to illustrate them using a particular example. Let's consider the Timed Petri net of Fig. 3. Its initial marking at the date $v_1 = 0$ is given by: $\overrightarrow{e_m}(0) = (1,0,0,0,1,0,1,1)^{\acute{u}}, \overrightarrow{e_r}(0) = (0,0,0,0)^{\acute{u}}$.

We assume that t_1 is fired at the date $v_1 = 1$, then simultaneously t_2 and t_4 at the date $v_2 = 4$. We need to calculate the state reached at the date $v_3 = 6$. The details of the calculation are given in Fig. 7 and 8. The *physical sense* of the equations is explained below:

- The quantity $(\overrightarrow{e_r} - \Delta_v \cdot \overrightarrow{1_N})^+$ represents the *update* of the residual durations vector at the date $v + \Delta_v$, from its value $\overrightarrow{e_r}$ at the date v. The ``+'' operator allows to take into account only positive values. Moreover, if a transition t is still active at date $v + \Delta_v$, we have: $(\overrightarrow{e_r} - \Delta_v \cdot \overrightarrow{1_N})^s\Big|_t = 1$;
- The quantity $\sum_{t \in \mathrm{T}} d(t) \cdot \overrightarrow{\varphi'}\Big|_t \cdot \overrightarrow{e_t}$ represents the *new residual durations* coming from the execution of the firing sequence φ' at the date $v + \Delta_v$;
- Finally, the quantity $\overrightarrow{e_r}^s - (\overrightarrow{e_r} - \Delta_v \cdot \overrightarrow{1_N})^s$ represents the Parikh vector of the transitions, the firing of which ends at the date $v + \Delta_v$. This expression is made from the comparison between the Parikh vector of the transitions that *were pending* at the date v: vector $\overrightarrow{e_r}^s$, and the Parikh vector of the transitions that will be *still active* at the date $v + \Delta_v$: vector $(\overrightarrow{e_r} - \Delta_v \cdot \overrightarrow{1_N})^s$.

4.2.4 Mathematical programming model

Since proposition 7 has been formulated in a linear way, it allows to express the linear mathematical programming model given below.

Model 2 (TPN Integer Programming Model). *Let* (R, d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{e_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* $v_0 = 0$. *Let* $e_f = \left(\overrightarrow{e_{m_f}}, \overrightarrow{0_N}\right)$ *be a target state. Let* $V \in \mathrm{N}$. *The integer linear programming model* $\mathrm{TIP}(V)$ *is defined by:*

$$Minimize \sum_{i \in [\![0, V-1]\!]} \Delta_{v_i} \tag{25}$$

subject to:

$$\forall k \in [\![1, M]\!], \quad e_{m0k} = \overrightarrow{e_{m_0}}\Big|_k \tag{26}$$

$$\forall k \in [\![1, M]\!], \quad e_{mVk} = 0 \tag{27}$$

$$\forall j \in [\![1, N]\!], \quad e_{r0\,j} = \overrightarrow{e_{r_0}}\Big|_j \tag{28}$$

$$\forall j \in [\![1, N]\!], \quad e_{rVj} = 0 \tag{29}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad B \cdot a_{ij} - e_{rij} \leq B - 1 \tag{30}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad e_{rij} - B \cdot a_{ij} \leq 0 \tag{31}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad B \cdot \alpha_{ij} - e_{rij} + \Delta_{v_i} \leq B - 1 \tag{32}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad e_{rij} - \Delta_{v_i} - B \cdot \alpha_{ij} \leq 0 \tag{33}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad e_{rij} - \Delta_{v_i} - \beta_{ij} \leq 0 \tag{34}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \beta_{ij} - B \cdot \alpha_{ij} \leq 0 \tag{35}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \beta_{ij} + B \cdot \alpha_{ij} - e_{rij} + \Delta_{v_i} \leq B \tag{36}$$

$$\forall i \in [\![1, V]\!], \quad \forall k \in [\![1, M]\!], \quad e_{mik} - e_{m(i-1)k} + \sum_{c=1}^{N} C_{kc}^{-} \cdot \varphi_{ic} - \sum_{c=1}^{N} C_{kc}^{+} \cdot (a_{(i-1)c} - \alpha_{(i-1)c}) = 0 \tag{37}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad e_{rij} - \beta_{(i-1)\,j} - \overrightarrow{d}\,|_j \cdot \varphi_{ij} = 0 \tag{38}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \varphi_{ij} + \alpha_{(i-1)\,j} \leq 1 \tag{39}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \varphi_{ij} \in \{0,1\} \tag{40}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad a_{ij} \in \{0,1\} \tag{41}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \alpha_{ij} \in \{0,1\} \tag{42}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad \beta_{ij} \in \mathrm{N} \tag{43}$$

$$\forall i \in [\![1, V]\!], \quad \forall j \in [\![1, N]\!], \quad e_{rij} \in \mathrm{N} \tag{44}$$

$$\forall i \in [\![1, V]\!], \quad \forall k \in [\![1, M]\!], \quad e_{mik} \in \mathbb{N} \quad (45)$$

$$\forall i \in [\![0, V-1]\!], \quad \Delta_{v_i} \in \mathbb{N} \quad (46)$$

Equations (26) to (29) correspond to conditions over initial and final states. Equations (30) to (36) express the constraints over discrimination variables used to compute the ``+'' and ``s'' operators. Variables (a_i), (α_i) and (β_i) denote respectively the values of $\overrightarrow{e_r}(v_i)^s$, $(\overrightarrow{e_r}(v_i) - \Delta_{v_i} \cdot \overrightarrow{1_n})^s$ and $(\overrightarrow{e_r}(v_i) - \Delta_{v_i} \cdot \overrightarrow{1_n})^+$ from equations (23) and (24). Equations (37) and (38) correspond to intermediate state computation equations (23) and (24). Equation (39) correspond to nonreentrance condition (21). Finally, equations (40) to (46) define the domain of the used variables.

Again, our model is well defined enough to allow the following proposition.

Proposition 8 (Correctness and completeness of TIP(v) w.r.t. $TP_1(v)$). *Let* (R, d) *be a TPN and* $v \in \mathbb{N}$ *. Then we have:*

- *Any solution of* $\mathrm{TIP}(v)$ *is also a solution of* $\mathrm{TP}_1(v)$ *(Correctness)*
- *Any solution of* $\mathrm{TP}_1(v)$ *can be expressed as a solution of* $\mathrm{TIP}(v)$ *(Completeness)*

Proof. Those results come directly from the construction of TIP(V). W

Obviously, the same remarks as for proposition 5 hold. Even if problem TP_1 is parametrized by a given number of timed steps , a large class of scheduling problems can be adressed using such formulation. We are more particularly interested in *flexible manufacturing systems* (FMS) scheduling problems. FMS are characterized by the *simultaneous* production of several types of products, and the possibility to use *several methods* (flexibilities) to produce the same kind of product. Using TPN, such flexibilities are modeled by *conflicts*, which justifies the use of our approach.

Another interest in the framework of FMS is the formulation of *cyclic* scheduling problems in a smart way. Indeed: such scheduling problems correspond to reachability between the same states (Bourdeaud'huy and Korbaa, 2006). Using our approach, one can formulate a cyclic scheduling problem by considering a timed reachability problem between two *identical unknown states*. The corresponding model allows then not only to find the schedule but also the initial state within the cycle.

Note finally that our mathematical model remains valid to solve the reachability problem between states defined by *not null* residual durations. One has just to consider that these states belong to a bigger problem between states without residual durations.

4.2.5 Numerical experiments

In order to validate the model above, preliminary experiments were carried out using the linear programming solver CPLEX 9.0. (Bourdeau'huy et al., 2006). They have shown promising results, but need to be extended in order to assess the efficiency of our approach compared to concurrent approaches from Operations Research litterature.

We also propose to develop cutting techniques allowing to improve the resolution performances. For instance, we suggest to reuse the decomposition technique described above. A *preliminary resolution* of the reachability problem between the initial and final state vectors in the underlying P/T Petri net can be used to obtain the Parikh vector of the firing sequence of the controlled execution searched for.

5. Conclusion and future work

In this chapter, we present techniques for solving *reachability problems* in PN and TPN based on *mathematical programming*. The approach is based on an *incremental search* using step sequences that represent parallel and reentrant firings of transitions. The mathematical model used allows the formulation and verification of *reachability-based* analysis problems.
Concerning PNs, we have proposed two formulations of the reachability problem, leading to integer and/or binary programming models. For each of them, we have developped some additional procedures, relaxation techniques and objective functions in order to improve the computational efficiency of the resolution. Numerical experiments have demonstrated the efficiency of our approach compared to standard ones from Artificial Intelligence and Petri nets community.
Several promising tracks will be considered in the future, such that:

- To develop rules to adjust dynamically the amplitudes of jump search, for example by exploiting *information* from the previous iterations and/or from the structure of the considered PN;
- To use heuristic methods to speed up the search or find a good bound on K_{min}.

Concerning TPN, we have shown how a linear integer programming model could be developped to solve the Timed Petri net reachability problem. This model is very general since it allows to deal with *weighted Timed* Petri nets, without restricting ourselves to an immediate firing semantic or Timed Event Graphs as it is done in the litterature. It can thus be directly used on flexible manufacturing models.
In the future, we propose to compare our computational results with concurrent approaches dedicated to scheduling problems. We also propose to develop cutting techniques allowing to improve the resolution performances.
Finally, we are currently adapting our incremental approaches to *Time Petri nets*, in order to be able to model scheduling problems with *Time Windows* associated to the tasks.

6. References

Baccelli, F., Cohen, G., Olsder, G., and Quadrat, J.-P. (1992). *Synchronization and linearity : An algebra for Discrete Event Systems*. Wiley, New York

Benasser, A. (2000). L'accessibilité dans les réseaux de Petri : une approche basée sur la programmation par contraintes. *PhD thesis*, Université des sciences et techologies de Lille

Benasser, A. and Yim, P. (1999). Railway Traffic Planning with Petri nets and Constraint Programming. *JESA*, 33(8-9), pp. 959–975

Berthelot, G. (1986). Transformations and Decompositions of Nets. *Advances in Petri Nets 1986 Part I, Proceedings of an Advanced Course*, Vol. 254, pp. 359–376

Berthomieu, B. and Diaz, M. (1991). Modeling and Verification of Time Dependent Systems using Time Petri Nets. *IEEE Trans. on Software Eng.*, 17(3), pp. 259–273

Bourdeaud'huy, T. (2004). Techniques d'Abstraction pour l'Analyse et la Synthèse de Réseaux de Petri. *PhD thesis*, Ecole Centrale de Lille

Bourdeaud'huy, T., Hanafi, S., and Yim, P. (2004a). Efficient Reachability Analysis of Bounded Petri nets using Constraint Programming. *SMC'04, International Conference on Systems, Man and Cybernetics,* La Hague, Hollande

Bourdeaud'huy, T., Hanafi, S., and Yim, P. (2004b). Recherche de Séquences d'Accessibilité dans les Réseaux de Petri utilisant l'Abstraction Logique et une réduction fondée sur l'équation d'état. *CIFA'04, Conférence Internationale Francophone d'Automatique*, Douz, Tunisie

Bourdeaud'huy, T., Hanafi, S., and Yim, P. (2004c). Solving the Petri Nets Reachability Problem using the Logical Abstraction Technique and Mathematical Programming. *CPAIOR'04, International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, LNCS 3011, pp. 112–127, Nice, France

Bourdeaud'huy, T., Hanafi, S., and Yim, P. (2006). Scheduling of Flexible Manufacturing Systems using Timed Petri nets and Mathematical Programming. *WODES'06, Workshop on Discrete Event Systems*, Ann Arbor, Michigan, USA

Bourdeaud'huy, T., Hanafi, S., and Yim, P. (2007). Mathematical Programming Approach for the Petri Nets Reachability Problem. *EJOR*, 177(1)

Bourdeaud'huy, T. and Korbaa, O. (2006). A Mathematical Model for Cyclic Scheduling with Work-In-Progress Minimization. *INCOM'06, IFAC Symposium on Information Control Problems in Manufacturing*, Saint Etienne, France

Briand, C. (1999). Solving the Car-Sequencing Problem using Petri Nets. *International Conference on Industrial Engineering and Production Management*, Vol. 1, pp. 543–551

Chrétienne, P. (1984). Exécutions Contrôlées dans les Réseaux de Petri Temporisés. *T.S.I.*, 3

Colom, J. and Silva, M. (1989a). Convex Geometry and Semiflows in P/T nets: a Comparative Study of Algorithms for Computation of Minimal P-semiflows. *Proceedings of the 10th International Conference on Application and Theory of Petri Nets*

Colom, J. and Silva, M. (1989b). Improving the Linearly based Characterization of P/T nets. *Proceedings of the 10th International Conference on Application and Theory of Petri nets*, pp. 52–73, Bonn, Germany

David, R. and Alla, H. (1992). Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems. Prentice-Hall

Fernandez, J.-C., Jard, C., Jéron, T., andMounier, L. (1992). ``On the fly" Verification of Finite Transition Systems. *Formal Methods in System Design*

Geoffrion, A. (1974). Lagrangean Relaxation for Integer Programming. *Mathematical Programming Study*, Vol. 2, pp. 82–114

Glover, F. (1977). Heuristics for Integer Programming using Surrogate Constraints. *Decision Sciences*, Vol. 8, pp. 156–166

Gunnarsson, J. (1998). Symbolic Tools for Verification of Large Scale DEDS. *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98)*, 11-14 October 1998, San Diego, CA, pp. 722–727.

Hillion, H. and Proth, J. (1989). Performance Evaluation of Job-Shop Systems using Timed Event Graphs. *IEEE Transactions on Automatic Control*, Vol. 34

Huber, P., Jensen, A. M., Jepsen, L. O., and Jensen, K. (1985). Towards Reachability Trees for High-level Petri Nets. *Lecture Notes in Computer Science: Advances in Petri Nets 1984*, Vol. 188, pp. 215–233

Jaffar, J., Michaylov, Stuckey, P., and Yap, R. (1992). The CLP (R) Language and System. *ACM Transactions on Programming Languages and Systems*, Vol. 14(3), pp. 339–395

Janicky, R. and Koutny, M. (1991). Optimal Simulations, nets and Reachability Graphs. *Advances in Petri Nets, Lecture Notes In Computer Science*, Vol. 524, pp. 205–226

Jensen, K. (1992). Coloured Petri nets - Basic Concepts, Analysis Methods and Practical Use. *EATCS Monographs on Theoretical Computer Science*, Vol. 1, pp. 1–234. Springer

Keller, R. (1976). Formal Verification of Parallel Programs. *Comm. of the ACM*, Vol. 19(7), pp. 371–384

Khomenko, V. and Koutny, M. (2000). Verification of Bounded Petri Nets using Integer Programming, *technical report cs-tr-711*, Department of Computing Science, University of Newcastle upon Tyne

Kosaraju, S. R. (1982). Decidability of Reachability in Vector Addition Systems. *Proc. Of the 14th Annual ACM Symp. on Theory of Computing*, pp. 267–281

Latvala, T. (2001). Model checking LTL Properties of High-level Petri Nets with Fairness Constraints. *Lecture Notes in Computer Science 2075*

Lautenbach, K. (1987). Linear Algebraic Techniques for P/T Nets. *Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course*, Vol. 254, pp. 142–167

Lee, D. Y. and DiCesare, F. (1994). Scheduling Flexible Manufacturing Systems using Petri nets and Heuristic Search. *IEEE Transactions on Robotics and Automation*, Vol. 10(2), pp. 123–132

Lindqvist, M. (1993). Parameterized Reachability Trees for Predicate/Transition Nets. *Lecture Notes in Computer Science; Advances in Petri Nets 1993*, Vol. 674, pp. 301–324

Lipton, R. (1976). The Reachability Problem requires Exponential Space. *Technical report*, Computer Science Dept., Yale University

Melzer, S. and Esparza, J. (1996). Checking System Properties via Integer Programming. *ESOP'96*

Murata, T. (1989). Petri Nets : Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol. 77, pp. 541–580

Parker, R. and Rardin, R. (1988). *Discrete Optimization*. Academic Press

Ramchandani, C. (1974). Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. *PhD thesis*, Massachusetts Institute of Technology

Richard, P. (2000). Modelling Integer Linear Programs with Petri nets. *RAIRO/Operations Research*, Vol. 34(3), pp. 305–312

Roch, S. and Starke, P. (2002). *INA Manual,* Integrated Net Analyzer, Version 2.2. Humboldt- Universität zu Berlin, Institut für Informatik

Schmidt, K. (1998). On the new Low Level Symmetry tool in INA. *GI Petri Net Newsletter 54*

Sifakis, J. (1975). Performance Evaluation of Systems using Nets. *Advanced Course: Net Theory and Applications*, pp. 307–319

Silva, F., Castilho, M. A., and Kunzle, L. A. (2000). Petriplan: A new Algorithm for Plan Generation (preliminary report). *IBERAMIA-SBIA*, pp. 86–95

Silva, M., Colom, J., and Campos, J. (1992). Linear Algebraic Techniques for the Analysis of Petri Nets. *Recent Advances in Mathematical Theory of Systems, Control, Networks, and Signal Processing II*

Silva,M., Teruel, E., and Colom, J.M. (1998). Linear Algebraic and Linear Programming Techniques for the Analysis of P/T Net Systems. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, Vol. 1491, pp. 309–373

Valmari, A. (1991). Stubborn Sets for Reduced State Space Generation. *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, Vol. 483, pp. 491–515

Van der Aalst,W.M. P. (1995). Petri Net Based Scheduling. Number 95. *Eindhoven University of Technology Computing Science Reports*/23

Vernadat, F., Azéma, P., and Michel, P. (1996). Covering Steps Graphs. *17th Int. Conf on Application and Theory of Petri Nets 96*.

Wang, J. (1998). *Timed Petri Nets, Theory and Application*. Kluwer Academic Publishers

19

Using Transition Invariants for Reachability Analysis of Petri Nets

Alexander Kostin
Eastern Mediterranean University
N.Cyprus

1. Introduction

Petri nets are an important formal paradigm for modeling and analysis of discrete event systems. The related areas of application of Petri nets include deadlock avoidance and prevention, supervisory control, forbidden state detection, different aspects of flexible manufacturing systems, and many others (Zhou & DiCesare, 1993; Holloway et al., 1997; Boel et al., 1995). Quite often, given a discrete-event system, the designer is interested in determining whether the system can transit from an initial state to another, target state as a result of some operations from a predefined set. In terms of Petri nets, the answer to this question is obtained as a solution of a *reachability problem*.

The reachability problem in Petri nets is formulated as follows: for any Petri net PN, with an initial marking M^0, and for some other marking M, determine whether the relation $M \in R(PN, M^0)$ is true, where $R(PN, M^0)$ is the reachability set of PN for its initial marking M^0 (Murata, 1989). The decidability of the reachability problem has been proved for a number of restricted classes of Petri nets, and there are efficient algorithms for such classes as acyclic Petri nets, marked graphs, and others (Kodama & Murata, 1988; Caprotti et al., 1995; Kostin, 1997).

It has been shown that the reachability problem is decidable for generalized Petri nets as well (Mayr, 1984). The fundamental contribution of the paper (Mayr, 1984) is in proving that the reachability problem for generalized Petri nets is decidable. However, being highly important theoretically, the practical use of the algorithm described in that paper is limited. Actually, the algorithm creates a series of so called regular constrained refined graphs, each of which is a generalization of the basic coverability tree. As the author admits, the first refined graph would enumerate the whole reachability set of the given Petri net.

In practice, two different approaches are used most often to determine the reachability of a marking in Petri nets. The first approach is based on the creation and investigation of a complete or reduced reachability graph. The main drawback of this approach is a *state explosion problem*. A closely related technique is the use of *stubborn sets*. The main purpose of the stubborn sets technique is to choose, for each marking of the net, a set of transitions to fire that is large enough to preserve some desired information about the Petri net, but is as small as possible to get a significant reduction of the resulting reachability graph (Varpaaniemi, 1998). Unfortunately, generation of minimal or reduced reachability graphs in finite state systems is known to be an NP-hard problem (Peled, 1993). If Petri net has no

specific properties like a symmetry or reversibility, the corresponding reduced reachbility graph will have almost the same size as that of the full reachability graph (Schmidt, 2000).
The second approach is based on methods of linear algebra. Given a *pure* Petri net (i.e. a net without self-loops), with sets of transitions T and places P, its structure is represented unambiguously by the *incidence matrix*

$$D = [d(t_i, p_j)] = [d_{ij}], \quad i = 1, 2, \ldots, m = |T|, \quad j = 1, 2, \ldots, n = |P|, \tag{1}$$

where $d(t_i, p_j) = Post(p_j, t_i) - Pre(p_j, t_i)$, Pre and $Post$ are the input and output functions of the Petri net, with $Pre(p, t) = v$ if there is a directed arc from p to t with the weight v, and $Post(p, t) = v$ if there is an arc from t to p with the weight v. Note that, in this matrix, rows correspond to transitions and columns correspond to places (Murata, 1989).
It is known that a necessary condition for reachability of marking M from some other marking M^0 is the existence of a nonnegative integer solution of the matrix equation

$$M = M^0 + FD \tag{2}$$

relative to F, where $F = [f_1, f_2, \ldots, f_m]$ is a nonnegative integer firing count vector. Note that in this chapter, all vectors are considered as row vectors. In particular, markings of PN will be expressed as $(1 \times n)$ vectors, so that we can write

$$M^0 = [m_1^0, m_2^0, \ldots, m_n^0] \quad \text{and} \quad M = [m_1, m_2, \ldots, m_n], \tag{3}$$

where the ith entry in vectors M^0 and M denotes the number of tokens in place $p_i \in P$.
Equation (2), proposed in (Murata, 1977), is called the *fundamental equation* of Petri net. It is of the paramount importance for the investigation of the structural and behavioral properties of Petri nets with methods of linear algebra.
With the use of linear algebra, rachability analysis is usually carried out in two stages. At the first stage, by solving the equation (2) or its related integer programming form, firing count vectors are obtained. At the second stage, the computed firing count vectors are used in an attempt to determine *legal firing sequences* that transform initial marking M^0 into target marking M.
Unfortunately, the existence of a nonnegative integer solution of equation (2) is not a sufficient condition for reachability of marking M from M^0 (Murata, 1989). That is, it is quite possible that, in a given Petri net, no legal firing sequences exist for the valid firing count vectors. In general, the equation (2) can have *infinite* number of nonnegative integer solutions. Some of these solutions can correspond to legal firing sequences, while others fail (Peterson, 1981). Thus, there is a challenging problem to select working firing count vectors.
In (Kostin, 2003), with the use of linear algebra, a method was proposed to restrict the number of firing count vectors to be tried for the determination of legal firing sequences, without the loss of reachability information. The method is applicable for reachability analysis of a particular class of place/transition Petri nets having no transition invariants, or T-invariants. Algebraically, T-invariants of a Petri net with incidence matrix D are nonnegative integer $(1 \times m)$ vectors F such that $FD = 0$ (Memmi & Roucairol, 1980). According to the scheme proposed in (Kostin, 2003), given a Petri net with an initial and a target markings, a so called *complemented* Petri net is created that consists of the given Petri net and an additional, *complementary* transition with some input and output places of the original Petri net, which are uniquely determined by the initial and target markings. Then

the reachability problem is reduced to computation and investigation of T-invariants of the complemented Petri net. The main result of that paper is that legal firing sequences, if they exist, can be found using only those T-invariants of the complemented Petri net in which the complementary transition fires only once. It was shown that this set is finite. This chapter generalizes the approach described in (Kostin, 2003) for arbitrary place/transition nets, including Petri nets with T-invariants. The existence of T-invariants in the original Petri nets considerably complicates the reachability analysis. In contrast with the scheme in (Kostin, 2003), where the number of T-invariants of any complemented Petri net that are sufficient for performing the reachability analysis is proven to be finite, in the generalized scheme the set of T-invariants for investigation is theoretically infinite. Nevertheless, as will be shown in this chapter, it is always possible to effectively limit this set without the loss of reachability information and then to use T-invariants from this finite set for reachability analysis.

This chapter is an extended version of the author's article published in Lecture Notes in Computer Science (Kostin, 2006). The use of the material of that article is done with kind permission of Springer Science and Business Media. The rest of the chapter is organized as follows. In Section 2, notation and basic statements used in the chapter are given. Section 3 explains how to compute so called minimal singular T-invariants of the complemented Petri net. In Section 4, a relation graph of T-invariants is introduced. Section 5 describes realization of T-invariants with borrowing of tokens. In Section 6, a scheme for linear combining of T-invarians is given. Section 7 illustrates the scheme by two examples. The most important points in sections are put down as proven statements. Some of the proofs are just skeletons or, for simple statements, omitted altogether.

2. Notation and basic statements

We adopt here the notation and basic statements from (Kostin, 2003). It is assumed without losing generality that Petri nets are *pure*, i.e. they have no self-loops. As was stated in the previous section, the structure of any pure Petri net is unambiguously represented by the incidence matrix (1).

Let M^0 be an initial marking and M be some other marking of given Petri net PN. If we are interested in reachability of M from M^0 then marking M will be called the *target* marking. It is assumed, throughout the chapter, that $M^0 \neq M$.

If marking M is reachable from marking M^0 in a Petri net PN, then there exists at least one sequence of markings $\mu = M^0 M^1 \dots M^r$ with $M^r = M$, and a legal firing sequence $\tau = t_{i_1} t_{i_2} \dots t_{i_r}$, with the two sequences related by the state equation $M^k = M^{k-1} + e[i_k]_m D$, $k = 1, 2, \dots, r$. Here $e[i_k]_m$ is an $(1 \times m)$ *control vector*, in which m - 1 entries are zero and the i_kth entry is one, indicating that a transition t_{i_k} fires at step k. Sequences μ and τ can be combined in one mixed sequence of interrelated markings and firing transitions that is called a *reachability path* from marking M^0 to marking M^r:

$$M^0 \xrightarrow{t_{i_1}} M^2 \xrightarrow{t_{i_2}} \dots \xrightarrow{t_{i_r}} M^r . \tag{4}$$

Its determination is the main problem of reachability analysis. As was stated in Section 1, with linear algebra methods, this analysis is usually carried out in two stages. At the first

stage, it is important to limit the number of firing count vectors, without the loss of reachability information. In the proposed approach, this stage is done with the use of T-invariants of so called complemented Petri net which is a simple extension of the original net .

Definition 1. For any Petri net *PN* with incidence matrix *D* specified by (1), and initial and target markings M^0 and M represented by vectors (3), there exists a unique *complemented* Petri net PN_c that has the same set of places *P* as *PN*, the set of transitions $T_c = T \cup \{t_{m+1}\}$, and is described structurally by the incidence matrix

$$D_c = \begin{bmatrix} D \\ \Delta M \end{bmatrix}, \tag{5}$$

where t_{m+1} is an additional, *complementary* transition, and $\Delta M = M^0 - M = [\Delta m_1, \Delta m_2, \ldots, \Delta m_n]$, with $\Delta m_i = m_i^0 - m_i$, $i = 1, 2, \ldots, n$ (Kostin, 2003). ♦

Using the right side of equation (2) with marking *M* instead of M^0, control vector $e[m + 1]_{m+1}$ instead of *F* and incidence matrix D_c instead of *D*, one can obtain

$$M + e[m + 1]_{m+1} D_c = M + \Delta M = M^0 . \tag{6}$$

That is, a *single* firing of the complementary transition in marking *M* of PN_c results in marking M^0.

It is known that the reproducibility of a firing sequence in a Petri net indicates the existence of a T-invariant (Memmi & Roucairol, 1980). Thus the following statement holds.

Statement 1. Given a Petri net *PN* with an initial marking M^0, a necessary condition for reachability of some other marking *M* is the existence of a T-invariant of the complemented Petri net PN_c, with a *single* firing of the complementary transition.

Denote by $F_c = [f_1, f_2, \ldots, f_m, f_{m+1}]$ a firing count vector of the complemented Petri net PN_c. Now Statement 1 may be reformulated as follows: given a Petri net *PN* with the incidence matrix *D* and an initial marking M^0, a necessary (but generally not sufficient) condition for some other marking *M* to be reachable from M^0 is the existence of an integer solution of the matrix equation

$$F_c D_c = 0 \tag{7}$$

relative to F_c, such that $F_c \geq 0$ and $f_{m+1} = 1$. Here D_c is the incidence matrix of PN_c as defined by (5). ♦

In sequel, each T-invariant of the complemented Petri net PN_c having the last entry $f_{m+1} = 1$ will be called a *singular complementary* T-invariant.

The importance of Statement 1 is that the reachability analysis of the *original* Petri net *PN* can be reduced to the computation and investigation of T-invariants of the complemented Petri net PN_c. One advantage of this reduction is the existence of efficient techniques for the calculation of T-invariants (Alaiwan, 1985; Krukeberg & Jaxy, 1987; Silva & Colom, 1991; Takano et al., 2001). Algorithms for the calculation of T-invariants are implemented in many Petri net software tools such as INA (Roch & Starke, 2001); GreatSPN (Chiola et al., 1995), TimeNET (German et al., 1995), and QPN (Bause & Kemper, 1994), to mention only a few.

It is known that, in any Petri net with T-invariants, there are *minimal-support T-invariants* which can be used as generators of all T-invariants of the given net (Memmi & Roucairol, 1980; Murata, 1989). Let $\Phi = \{F_1, F_2, \ldots, F_s\}$ be the set of minimal-support T-invariants of some Petri net consisting of $m = |T|$ transitions, where $F_i = [f_{i1}, f_{i2}, \ldots, f_{im}] \geq_{\neq} 0$, and s is the number of minimal-support T-invariants. We use here, for a vector X, a denotation $X >_{\neq} 0$ if $X \geq 0$ and $x_i \neq 0$ for some ith entry of X. Each $F_i \in \Phi$ specifies a nonempty subset of transitions $\|F_i\| \subseteq T$ such that $t_j \in \|F_i\|$ if and only if $f_{ij} > 0$, with $\|F_i\| \not\subset \|F_k\|$ and $\|F_k\| \not\subset \|F_i\|$ for every pair of distinct indices $i, k = 1, 2, \ldots, s$. Here $\|F_i\|$ represents the *minimal support* of T-invariant F_i.

Statement 2. In any Petri net the number of minimal-support T-invariants is finite (Kostin, 2003).

Statement 3. For any Petri net *PN*, its complemented net PN_c includes all T-invariants of *PN* (Kostin, 2003).

Statement 4. For every reachability path from an initial marking M^0 to a target marking M of a given Petri net *PN*, there exists a T-invariant $F = [f_1, f_2, \ldots, f_m, f_{m+1}]$ of the corresponding complemented Petri net PN_c, with $f_{m+1} = 1$. That is, F is a singular complementary T-invariant of PN_c.

Let

$$M^0 \xrightarrow{t_{i_1}} M^1 \xrightarrow{t_{i_2}} \ldots \xrightarrow{t_{i_k}} M^k = M \tag{8}$$

be some reachability path from M^0 to M in given Petri net *PN*, such that $M^j \neq M$ and $t_{i_j} \neq t_{m+1}$ for $j = 1, 2, \ldots, k\text{-}1$. Using this path, create an *expanded* reachability path

$$M^0 \xrightarrow{t_{i_1}} M^1 \xrightarrow{t_{i_2}} \ldots \xrightarrow{t_{i_k}} M^k \xrightarrow{t_{i_{k+1}}} M^{k+1} = M^0. \tag{9}$$

Since $M^k = M$, marking M^k can be transformed, according to (6), into marking M^0 by a single firing of the complementary transition $t_{i_{k+1}} = t_{m+1}$. Consider now the firing count vector corresponding to the reachability path (9):

$$F = [f_1, f_2, \ldots, f_m, f_{m+1}], \tag{10}$$

where f_i is the number of times transition t_i appears in the sequence $t_{i_1} t_{i_2} \ldots t_{i_{k+1}}$, with $f_{m+1} = 1$. Since, in the reachability path (9), initial marking M^0 is transformed back into M^0, the corresponding firing count vector (10) is a T-invariant. Further, since the last entry in this vector $f_{m+1} = 1$, the vector is a singular complementary T-invariant of the complemented Petri net PN_c. ♦

Note that the reverse of Statement 4 is generally not true. That is, the existence of a singular complementary T-invariant does not guarantee that there exists a corresponding reachability path.

Corollary 1. For any Petri net, with given initial and target markings M^0 and M respectively, all existing reachability paths from M^0 to M are the paths that can be created on the set of singular complementary T-invariants. This corollary is a generalization of the corresponding result for T-invariant-less Petri nets obtained in (Kostin, 2003). It means that,

to perform reachability analysis of a Petri net, it is sufficient to search for reachability paths only on the set of singular complementary T-invariants. ♦

The set implied by Corollary 1 is infinite in general and includes singular minimal-support complementary T-invariants and all linear combinations of minimal-support T-invariants that yield the last entry $f_{m+1} = 1$. As will be shown, it is sufficient to consider in this set, without losing reachability information, only a finite subset.

Let

$$\Phi_c = \{F_1, F_2, \ldots, F_w\} \tag{11}$$

be a set of all minimal-support T-invariants of PN_c, where

$$F_j = [f_{j1}, f_{j2}, \ldots, f_{jm}, f_{j,m+1}] \geq_{\neq} 0, \tag{12}$$

with $j = 1, 2, \ldots, w$. Notice that, according to the basic property of a T-invariant, each entry in vector F_j may be only a nonnegative integer (Memmi & Roucairol, 1980).

Now, depending on the value of the last entry, the minimal-support T-invariants of set Φ_c can be classified into the following three *disjoint* groups:

$$\{F_j \mid f_{j,m+1} = 0,\ j \in I_w\}, \tag{13}$$

$$\{F_j \mid f_{j,m+1} = 1,\ j \in I_w\}, \tag{14}$$

$$\{F_j \mid f_{j,m+1} > 1,\ j \in I_w\}. \tag{15}$$

where $I_w = \{1, 2, \ldots, w\}$ is the indexing set of Φ_c. According to Statement 2, each of these groups is finite. Depending on the Petri net and its initial and target markings, some or even all these three groups can be empty.

Without the last, $(m+1)$th entry, T-invariants of group (13), by Statement 3, are minimal-support T-invariants of the original Petri net *PN*. We will call members of group (13) *non-complementary* minimal-support T-invariants of the complemented Petri net PN_c. Group (14) consists of singular complementary T-invariants. Finally, members of group (15) are nonsingular complementary T-invariants in which the complementary transition fires more than once. Together, members of groups (14) and (15) are called minimal-support *complementary* T-invariants of PN_c.

3. Computing minimal singular T-invariants of a complemented Petri net

By Corollary 1, the search for all reachability paths from initial marking M^0 to target marking M in a given Petri net can be carried out only on singular T-invariants of the corresponding complemented Petri net. These include, first of all, minimal-support T-invariants of group (14). However, these are not the only singular T-invariants of the complemented Petri net. Indeed, linear combinations of minimal-support T-invariants of groups (13), (14), and (15) can yield additional singular T-invariants. The number of such combinations is infinite in general. In this section, we will show that there exists a finite set of *minimal* singular T-invariants of the complemented Petri net. Then an approach to the computation of such a set will be described. In Section 6, it will be shown how the

computed minimal singular T-invariants can be combined with non-complementary T-invariants of group (13) to produce new, non-minimal singular T-invariants.
Consider a linearly-combined T-invariant

$$F = [f_1, f_2, ..., f_m, f_{m+1}] = \sum_{j=1}^{w} k_j F_j \tag{16}$$

with rational coefficients k_j, where F_j are minimal-support T-invariants of groups (13), (14) and (15), and w is the number of elements in the three groups. In agreement with Corollary 1, we are looking only for those combined T-invariants F which yield $f_{m+1} = 1$. Thus, the following constraint must hold for each linear combination F in (16):

$$f_{m+1} = \sum_{j=1}^{w} k_j f_{j,m+1} = 1. \tag{17}$$

With $k_j \geq 0$, the product $k_j F_j$ in (16) can be considered as a contribution of firings of transitions of T-invariant F_j to firings of transitions of the combined T-invariant F. On the other hand, a negative coefficient k_j in (16) may be interpreted as a reverse, or backward firing of transitions, corresponding to T-invariant F_j, and this is *not legal* in the normal semantics of Petri nets. Thus, for T-invariants of groups (14) and (15), taking into account (17), their coefficients k_j must be in the following range:

$$0 \leq k_j \leq 1. \tag{18}$$

That is, for groups (14) and (15), in which $f_{j,m+1} \geq 1$, to satisfy (17) the following inequality must hold:

$$\sum k_j \leq 1. \tag{19}$$

However, coefficients k_j for T-invariants of group (13) in (16) may have arbitrary (non-negative) values without affecting the constraint (17). As a particular case, these T-invariants can be combined in (16) with coefficients $k_j \leq 1$. The case when T-invariants of group (13) can be included into combination (16) with arbitrary large coefficients is considered in Section 6.
The linearly-combined T-invariants (16), with the constraints (17), (18) and (19), are called *minimal singular* T-invariants of the complemented Petri net. As a subset, they include all minimal-support T-invariants of group (14).
Minimal singular T-invariants of the complemented Petri net can be computed in the following way. Rewrite (16) as a system of linear algebraic equations

$$\Psi K^T = F^T, \tag{20}$$

where Ψ is a matrix of size $((m + 1) \times w)$ whose columns are transposed minimal-support T-invariants F_j from groups (13), (14) and (15), $K = [k_1, k_2, \ldots, k_w]$, and F is vector (16), with $f_{m+1} = 1$.
In the system (20), not only coefficient vector K, but also entries f_i of F, for $i = 1, 2, \ldots, m$, are not known. We will show, however, that the number of different integer-valued vectors F with $f_{m+1} = 1$ is finite. Then we will explain how to compute the valid vectors in (20). The word "valid" means here that, in addition to the requirement $f_{m+1} = 1$, all coefficients k_j in

(16) satisfy the constrains (18) and (19). Taking into account (17) and (18), one can deduce that

$$0 \le f_i \le \max_j(f_{ji}),\ \ j = 1,\ 2,\ ...,\ w;\ \ i = 1,\ 2,\ ...,\ m, \tag{21}$$

where entries f_i are integer-valued components of vector F in (16).
One can see now that the number of different integer-valued vectors F in the system (20) is

$$N = \prod_{i=1}^{m} [\max_j(f_{ji}) + 1]. \tag{22}$$

This number includes one vector F with all zero entries except the last one, and all minimal-support T-invariants of group (14). Among the remaining vectors F, there can be additional singular T-invariants. They can be computed in the following way.
Assume that, in the system (20), K is a vector of unknowns. Then Ψ can be considered as a coefficient matrix, so that the augmented matrix of the system (20) is $U = \Psi \mid F^T$. It is known that, by elementary row operations, each matrix can be transformed to an upper trapezoidal form (Goldberg, 1991). In particular, for the augmented matrix U the result of its transformation $U^{\sim}$ can be written as follows:

$$U^{\sim} = \begin{bmatrix} * & * & * & * & ... & * & y_1 \\ 0 & * & * & * & ... & * & y_2 \\ 0 & 0 & * & * & ... & * & y_3 \\ \circ & \circ & \circ & \circ & ... & \circ & \circ \\ \circ & \circ & \circ & \circ & ... & \circ & y_m \\ \circ & \circ & \circ & \circ & ... & \circ & y_{m+1} \end{bmatrix}, \tag{23}$$

where the symbol '*' stands for some value (this value is not zero if the symbol is the first in the row), the symbol 'o' is a place holder, and $y_i = y_i(f_1, f_2, \ldots, f_m, f_{m+1})$ is some linear function of its arguments, $i = 1, 2, \ldots, m+1$. Each row in $U^{\sim}$ consists of $w + 1$ elements.
For the system (20) to be consistent, the following equation must hold for each ith row of matrix $U^{\sim}$ with all w leading elements equal to zero (Goldberg, 1991):

$$y_i(f_1, f_2, \ldots, f_m, f_{m+1}) = 0. \tag{24}$$

Collecting now all equations (24), we obtain a derived system of linear algebraic equations

$$\begin{aligned} &y_{j_1}(f_1, f_2, ..., f_m, f_{m+1}) = 0 \\ &y_{j_2}(f_1, f_2, ..., f_m, f_{m+1}) = 0 \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ &y_{j_k}(f_1, f_2, ..., f_m, f_{m+1}) = 0 \end{aligned} \tag{25}$$

where $k \le m$ and $f_{m+1} = 1$.

Integer solutions of this system relative to $f_1, f_2, \ldots, f_m$ can be found using existing algorithms for integer systems of linear equations (Howell, 1971; Springer, 1986). With the constraints (21), the system has a finite number of solutions or no solutions at all. Note that, with nonempty group (14), for all its members $[f_{j1}, f_{j2}, \ldots, f_{jm}, 1]$, the system (25) has solutions at least for the trivial linear combinations

$$F = [f_1, f_2, \ldots, f_m, 1] = [f_{j1}, f_{j2}, \ldots, f_{jm}, 1], \tag{26}$$

since each vector (26) is the solution of (20), for which vector K has some entry $k_j = 1$, with all other coefficient entries equal to zero.

To illustrate this method, consider a Petri net of 6 transitions and 6 places having the incidence matrix

$$D = \begin{bmatrix} -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

with the initial and target markings $M^0 = [2, 0, 0, 0, 0, 0]$ and $M = [0, 0, 0, 0, 0, 2]$, respectively. The corresponding complemented Petri net has two minimal-support T-invariants $F_1 = [0, 0, 2, 2, 2, 0, 1]$ and $F_2 = [2, 2, 0, 0, 0, 2, 1]$. Both are *singular* T-invariants (that is, they have $f_{m+1} = f_7 = 1$). We will try to determine whether there are some other minimal singular T-invariants. For this example, with $w = 2$, the augmented matrix of the system (20) and its upper trapezoidal form are

$$\begin{bmatrix} 0 & 2 & f_1 \\ 0 & 2 & f_2 \\ 2 & 0 & f_3 \\ 2 & 0 & f_4 \\ 2 & 0 & f_5 \\ 0 & 2 & f_6 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & f_1 \\ 0 & 0 & f_1 - f_2 \\ 0 & 0 & f_3 - f_4 \\ 0 & 0 & f_4 - f_5 \\ 0 & 0 & f_1 - f_6 \\ 0 & 0 & f_1 + f_3 - 2 \end{bmatrix}.$$

Thus, the system (25) is

$$\begin{aligned} f_1 - f_2 &= 0 \\ f_3 - f_4 &= 0 \\ f_4 - f_5 &= 0 \\ f_1 - f_6 &= 0 \\ f_1 + f_3 &= 2 \end{aligned}$$

With the constraints $0 \le f_1, f_2, f_3, f_4, f_5, f_6 \le 2$, this system has the following three nonnegative integer solutions: [0, 0, 2, 2, 2, 0, 1], [2, 2, 0, 0, 0, 2, 1] and [1, 1, 1, 1, 1, 1, 1]. Clearly, the fist two solutions are minimal-support T-invariants F_1 and F_2, and the third solution is a minimal singular T-invariant that is the linear combination $F_3 = 0.5F_1 + 0.5F_2$. Neither F_1 nor F_2 are realizable in given initial marking. However, their linearly combined T-invariant F_3 is realizable. One legal firing sequence is $t_3\, t_1\, t_2\, t_4\, t_5\, t_6\, t_7$.

4. Relation graph of T-invariants

In general, each singular T-invariant should be tested for the creation of a reachability path (or a legal firing sequence) not only *alone*, but also in different linear combinations with non-complementary T-invariants (13), since these T-invariants can "help" the singular T-invariant to become realizable in given initial marking M^0 and to eventually provide a reachability path from M^0 to a target marking M. As will be shown in this section, in general not all non-complementary T-invariants can affect realization of the given singular T-invariant.

Definition 2. Let F be a T-invariant of a Petri net, with the support $\|F\|$. Then

$$P(F) = \{p_j \mid t_i \in \|F\|, d_{ij} \neq 0\} \tag{27}$$

is a set of places of this Petri net *affected* by F when it becomes realizable in some marking. Here, d_{ij} is an element of the incidence matrix of the Petri net as specified by (1). ♦

Statement 5. Let F_1 and F_2 be some T-invariants of a Petri net, and let P_1 and P_2 be sets of places affected by F_1 and F_2 respectively. If $P_1 \cap P_2 = \varnothing$, then T-invariants F_1 and F_2 have no *direct* effect on the realizability of each other.

Assume that, contrary to the statement, F_1 can directly affect the realizability of F_2. This is possible only if F_1, during its realization, will change the number of tokens in some places affected by F_2. This can happen only if $P_1 \cap P_2 \neq \varnothing$. The contradiction proves the statement. ♦

Even if $P_1 \cap P_2 = \varnothing$, T-invariants F_1 and F_2 can *indirectly* affect the realizability of each other through other T-invariants having common affected places with F_1 and F_2.

Corollary 2. Let $F_{nc}^1, F_{nc}^2, \ldots, F_{nc}^k$ be some non-complementary T-invariants of a complemented Petri net, with sets of places $P_{nc}^1, P_{nc}^2, \ldots, P_{nc}^k$ affected by these T-invariants, respectively. Let further F_c be a singular complementary T-invariant of this Petri net, with the set of affected places P_c. Denote by $P_{nc} = \bigcup_{i=1}^{k} P_{nc}^i$ a set of places of this Petri net affected by mentioned *non-complementary* T-invariants.

If $P_c \cap P_{nc} = \varnothing$, then realization of any linear combination of T-invariants $F_{nc}^1, F_{nc}^2, \ldots, F_{nc}^k$ has no effect on realization of F_c. Therefore these T-invariants may be excluded from consideration in the reachability analysis with T-invariant F_c in given Petri net. ♦

To represent formally the effects of different T-invariants on each other in a Petri net, it is instructive to introduce into consideration a *relation graph* of T-invariants. Nodes in this graph are T-invariants. Two nodes corresponding to T-invariants F_i and F_j are connected by a non-oriented edge if $P(F_i) \cap P(F_j) \neq \varnothing$, and the corresponding T-invariants F_i and F_j are called *directly connected* T-invariants.

For a Petri net, such a graph generally consists of a number of connected components. A connected component may include complementary and non-complementary T-invariants, or only one type of T-invariants. We say that two T-invariants F_i and F_j can affect realizability of each other if they belong to the same connected component, even if $P(F_i) \cap P(F_j) = \varnothing$. On the other hand, if F_i and F_j belong to different connected components, they can not affect each other in no way, directly or indirectly.
The algorithm for determining all connected components of a graph is well known (Goodrich, 2002). In our problem, the algorithm will determine a connected component consisting of nodes representing a given singular T-invariant and non-complementary T-invariants. For this purpose, the algorithm will use the incidence matrix of the *original* Petri net and the array of T-invariants.

5. Realization of T-invariants with borrowing of tokens

In this section, the meaning of the help provided by one T-invariant to another one to become realizable is explained. Let p be a place affected by two T-invariants F_i and F_j in a given Petri net. Assume that, in a given initial marking of the net, F_i is realizable, but F_j can become realizable if place p accumulates r_j tokens during realization of T-invariant F_i. Suppose further that, at some intermediate step during realization of F_i, r_i tokens will be created in place p. If $r_i \geq r_j$ then, by temporary borrowing of r_j tokens in place p, T-invariant F_j becomes realizable and, at the end of its realization, will return the borrowed tokens to place p, so that T-invariant F_i can complete its started realization.
With $r_i < r_j$, T-invariant F_j cannot borrow the necessary number of tokens in place p. However, if T-invariant F_i, after creation of r_i tokens in p at some step of its first realization, can start a new realization before the completion of the first one, then additional r_i tokens will be created in place p, so that this place will now accumulate $2r_i$ tokens. In general, if F_i can start z realizations before the completion of the previous ones, then place p will accumulate zr_i tokens. If, for some z, $zr_i \geq r_j$ then, after borrowing r_j tokens in p, T-invariant F_j becomes realizable. After the completion of its realization, all tokens borrowed by F_j will be returned to place p, and T-invariant F_i can complete all its started realizations.

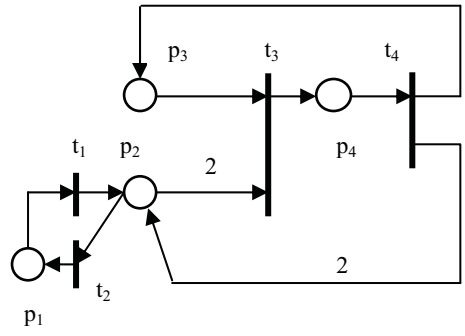


Fig. 1. Illustration of borrowing of tokens by a T-invariant.

Borrowing of tokens by a T-invariant is illustrated with a Petri net shown in Fig. 1, with arcs (p_2, t_3) and (t_4, p_2) having multiplicity 2. This net has two minimal-support T-invariants F_1 = [1, 1, 0, 0] and F_2 = [0, 0, 1, 1]. In the initial marking M^0 = [2, 0, 1, 0], F_1 is realizable, but F_2

becomes realizable only if it can borrow two tokens in place p_2, affected by the both T-invariants. These two tokens will be created here after T-invariant F_1 starts two realizations by firing transition t_1 two times. Afterwards, F_2 becomes realizable by borrowing two tokens in p_2. Then, after firing t_3 and t_4, the borrowed tokens reappear in p_2, and F_1 can complete its two started realizations. The corresponding sequence of transition firings for this example is $t_1t_1t_3t_4t_2t_2$.

To represent the relationship between *connected* T-invariants, when some non-realizable T-invariants can become realizable in given initial marking of a Petri net by borrowing tokens in places affected by other T-invariants, we will introduce a two-dimensional borrowing matrix G. In this matrix, rows correspond to T-invariants and columns correspond to places of the given Petri net. Formally, for a group of connected T-invariants,

$$G = [g_{ij}],\ i = 1, 2, \ldots, s;\ j = 1, 2, \ldots, n, \tag{28}$$

where s is the number of connected T-invariants in the group and n is the number of places in the net. The elements of matrix G are integers and have the following meaning. If $g_{ij} > 0$ then, for its realization, T-invariant F_i needs to borrow g_{ij} tokens in place p_j affected by some other T-invariant of the considered group. If $g_{ij} < 0$ then T-invariant F_i, at some intermediate step of its *single* realization, creates $|g_{ij}|$ tokens in place p_j. Finally, $g_{ij} = 0$ means that F_i does not affect place p_j.

As an example, matrix G for minimal-support T-invariants of the Petri net shown in Fig. 1 is:

	p_1	p_2	p_3	p_4
F_1	-1	-1	0	0
F_2	0	2	-1	-1

One can see from this matrix that the number of tokens created in place p_2 during a single realization of F_1 is 1 and is not sufficient for F_2 to borrow two tokens. In this example borrowing is possible if T-invariant F_1 starts two *interleaved* realizations. The maximal number of realizations that can be started by F_1 depends on the initial marking of place p_1. In particular, if this place initially contains only one token, then F_1 is still realizable, but it will never create, during its realizations, more than one token in p_2.

For a group of connected T-invariants of a complemented Petri net, the borrowing matrix can be created with the use of the incidence matrix of the given original Petri net. Due to a relative simplicity of the underlying procedure and to space limitation, the details of this procedure are omitted.

6. Combining a singular complementary T-invariant with non-complementary T-invariants

Denote by F_c a singular T-invariant of some complemented Petri net. It can be a member of group (14) or a minimal T-invariant calculated as was described in Section 3. Clearly, if group (13) is not empty, then the following linear combination

$$F = F_c + \sum k_j F_{nc}^j, \tag{29}$$

with coefficients $k_j \geq 0$, is also a singular T-invariant, if components of F are nonnegative integers. Here F_{nc}^{j} is a T-invariant of group (13). According to Corollary 2, it is sufficient to include in (29) only those T-invariants from (13) that belong to the same group of connected T-invariants together with F_c.

The expression (29) implies that the singular T-invariant F_c in general should be tested for the determination of a reachability path not only alone, but also in different linear combinations with non-complementary T-invariants (13), since these T-invariants can "help" the non-realizable T-invariant F_c to become realizable in given initial marking M^0 and to eventually provide a reachability path from M^0 to a target marking M of the given Petri net.

Without loosing generality, we assume that coefficients k_j in (29) are nonnegative integers. Indeed, if a singular T-invariant F_c is realizable with some non-integer values of coefficients k_j in (29), then it will remain realizable when these coefficient values are replaced by the nearest integer values not less than k_j. The case when $k_j \leq 1$ was considered in Section 3.

With integer coefficients $k_j > 1$, the product $k_j F_{nc}^{j}$ in (29) corresponds to a multiple realization of T-invariant F_{nc}^{j}. A multiple realization is a series of k_j sequential or interleaved single realizations. Interleaved realizations of a T-invariant, if they are possible, can have a different effect on place marking in comparison with sequential realizations. Consider, for example, a simple Petri net consisting of two transitions t_1, t_2 and one place p that is the output place for t_1 and the input place for t_2. This Petri net has a T-invariant F = [1, 1] realizable in any initial marking of p. In particular, with the zero initial marking, place p will never have more than one token if single realizations of F are strictly sequential as in $t_1t_2t_1t_2t_1t_2$. However, if single realizations of F are interleaved, place p can accumulate an arbitrary large number of tokens at some intermediate step.

In general, the number of valid combinations (29) is infinite. This section describes how to limit the values of coefficients k_j in (29) without the loss of reachability information using the concept of structural boundedness of Petri nets.

It is known (Murata, 1989) that a Petri net is *structurally bounded* if and only if there exists a (1 × n) vector $Y = [y_1, y_2, \ldots, y_n]$ of positive integers, such that

$$D\,Y^T \leq 0, \tag{30}$$

where D is the ($m \times n$) incidence matrix of the Petri net with m transitions and n places.

A Petri net is said to be *not structurally bounded* if and only if there exists a (1 × m) vector of (nonnegative) integers $X = [x_1, x_2, \ldots, x_m] \underset{\neq}{>} 0$, such that

$$D^T X^T = \Delta M^T \tag{31}$$

for some $\Delta M \underset{\neq}{>} 0$, where m is the number of transitions in the Petri net, and ΔM is a (1 × n) vector of marking increments as a result of firing of all transitions corresponding to vector X.

In a structurally unbounded Petri net, at least one place is structurally unbounded. A place p_i in such a Petri net is said to be *structurally unbounded* if and only if there exists a (1 × m) vector $X \underset{\neq}{>} 0$ of nonnegative integers, such that

$$D^T X^T = \Delta M^T \tag{32}$$

for some $\Delta m_i > 0$ in $\Delta M = (\Delta m_1, \Delta m_2, ..., \Delta m_i, ..., \Delta m_n) \underset{\neq}{>} 0$.

The structural unboundedness can be tested separately for each place p_i of the Petri net, by setting an appropriate integer $\Delta m_i > 0$ and $\Delta m_j = 0$ for all $j \neq i$ in (32) and then trying to solve the system (32). The test may be done also simultaneously for a few desired places or even for all places of the net.

It is known that, according to Minkowski-Farkas' lemma (Kuhn & Tucker, 1956), one of the systems (30) or (31) has solutions. For our problem, we do not need to know all solutions of (30) or (31). Rather, it is sufficient to find only one, "minimal" solution of (30) or (31).

The minimal solutions of (30) or (31) can be found as solutions of integer linear programming (ILP) problems. For the system (30), the corresponding ILP problem can be formulated as follows:

$$\text{minimize} \quad a = \sum_{i=1}^{n} y_i, \tag{33}$$

$$\text{subject to:} \quad DY^T \leq 0, \;\; y_i \geq 1, \;\; i = 1, \; 2, ..., n.$$

For the system (31), the corresponding ILP problem is:

$$\text{minimize} \quad b = \sum_{i=1}^{m} x_i, \tag{34}$$

$$\text{subject to:} \quad D^T X^T \underset{\neq}{>} 0, \;\; \sum_{i=1}^{m} x_i \geq 1, \;\; x_i \geq 0, \;\; i = 1, \; 2, ..., m.$$

The property of structural boundedness can be considered also for subnets of a Petri net. We are interested in this property only for the subnets corresponding to non-complementary T-invariants F_{nc}^j in (29). For a non-complementary T-invariant F_{nc}^j, the related subnet consists of transitions of the support $\| F_{nc}^j \|$ and places $P(F_{nc}^j)$ affected by F_{nc}^j. The expressions (30) - (34) remain valid for the subnet corresponding to F_{nc}^j with the following restrictions: in the incidence matrix D rows are taken for transitions corresponding to nonzero entries in F_{nc}^j, and columns are taken for places affected by F_{nc}^j.

Let us consider initially the case when the subnet corresponding to F_{nc}^j is not structurally bounded and describe how to determine coefficients k_j for non-complementary T-invariants F_{nc}^j in the linear combination (29). If F_{nc}^j and F_c belong to different connected

components of the graph of relation of T-invariants then F_{nc}^{j} should be ignored at all, by setting $k_j = 0$ in (29).

If F_{nc}^{j} and F_c belong to the same connected component of the graph of relation of T-invariants then the subnet corresponding to F_{nc}^{j} will have common places with the subnets corresponding to F_c or other non-complementary T-invariants belonging to the same connected component. Thus, F_{nc}^{j} can affect realizability of F_c, directly or indirectly, and therefore should be included in (29) with $k_j > 0$.

Suppose for definiteness that T-invariant F_{nc}^{j} has the support $\{t_1, t_2, \ldots, t_l\}$, $l \le m$, and the set of affected places

$$\{p_1, p_2, \ldots, p_q\}, \; q \le n, \tag{35}$$

where m and n are the numbers of transitions and places in the original (non-complemented) Petri net. Assume that F_c, to become realizable, needs to borrow $n_i > 0$, $i = 1, 2, \ldots, h$, tokens at least in places

$$\{p_1, p_2, \ldots, p_h\}, \quad h \le q, \tag{36}$$

that belong to the set (35) and in which F_{nc}^{j} can create tokens during its realization. Then, to facilitate the realizability of F_c, F_{nc}^{j} should be included in the linear combination (29) with a positive integer coefficient k_j determined by applying the following steps.

1. Try to solve an ILP problem:

$$\text{minimize} \quad b = \sum_{i=1}^{l} x_i, \tag{37}$$

$$\text{subject to:} \quad D^T X^T \ge \Delta M^T, \; \sum_{i=1}^{l} x_i \ge 1, \; x_i \ge 0,$$

where $\Delta M = [\Delta m_1, \Delta m_2, .., \Delta m_h, \Delta m_{h+1}, \ldots, \Delta m_q] = [n_1, n_2, \ldots, n_h, 0, \ldots, 0]$ is a vector of the desired numbers of tokens which are expected to be created in places (36) as a result of one or more realizations of F_{nc}^{j}, l is the number of transitions in the subnet corresponding to F_{nc}^{j}, and q is the number of places affected by F_{nc}^{j}. In the matrix multiplication, only those rows and columns of D are used which correspond to the support of F_{nc}^{j} and to places affected by F_{nc}^{j}.

2. If, for the specified vector ΔM, the problem (37) has a solution $X^* = [x_1^*, x_2^*, \ldots, x_l^*]$, then components of X^* represent the total numbers of firings of respective transitions *sufficient* to accumulate the desired number of tokens in places of set (36) in a few realizations of F_{nc}^{j}, and ratio $\left\lceil \frac{x_i^*}{f_i^j} \right\rceil$ is the number of realizations of F_{nc}^{j} to provide the necessary number of firings of transition t_i, $i = 1, 2, \ldots, l$. In this case,

$$k_j = \max\left(\left\lceil \frac{x_i^*}{f_i^j} \right\rceil \mid \; i = 1,\ 2,\ ...,\ l\right). \tag{38}$$

3. If, on the other hand, the problem (37) has no feasible solution then it means that at least one of places in set (36) p_i is structurally bounded and can not accumulate the desired number of tokens Δm_i in multiple realizations of F_{nc}^j. In this case, using (32), determine all structurally unbounded places in set (36). Since, as is assumed, the subnet for F_{nc}^j is not structurally bounded, there is at least one structurally unbounded place in this subnet.
4. Solve the ILP problem (37) simultaneously for all structurally unbounded places found at the previous step, to obtain a solution vector X^*. That is, in solving (37), vector ΔM should have nonzero entries $\Delta m_i = n_i$ only for structurally unbounded places. According to Minkowski-Farkas' lemma, this solution always exists. Then coefficient k_j is determined by the use of expression (38).

In case, when the subnet for F_{nc}^j is found to be structurally bounded, then the number of tokens in each of its places is bounded. However, this bound generally depends on realizations of other, connected T-invariants and is not known in advance. For such a subnet, coefficient k_j can be evaluated with the use of the borrowing matrix (28) computed for F_c and all its connected non-complementary T-invariants, including F_{nc}^j. Let, in this matrix, c and j be indexes of rows corresponding to F_{nc}^j and F_c, respectively. Then it is *sufficient* to include F_{nc}^j in the linear combination (29) with coefficient k_j computed with the use of the expression

$$k_j = \sum \left\lceil \frac{g_{ci}}{|\, g_{ji} \,|} \right\rceil, \tag{39}$$

where g_{ci} and g_{ji} are entries in the borrowing matrix, and the sum is computed for all pairs $g_{ci} > 0$ and $g_{ji} < 0$. Indeed, with this coefficient, the sufficient number of interleaved realizations of F_{nc}^j are *allowed* to accumulate the required numbers of tokens in places which are common for F_c and F_{nc}^j and in which T-invariant F_c can borrow them during its realization.

However, the possibility of realizations of F_{nc}^j depends on marking of places in its subnet. For example, in the Petri net of Fig. 1, T-invariant F_2 can become realizable only with the help of T-invariant F_1 for which the corresponding subnet is structurally bounded. The borrowing matrix for this example has only one pair of non-zero entries $g_{12} = -1$ (for F_1) and $g_{22} = 2$ (for F_2). Thus, using (39), one can obtain $k_1 = 2$. That is, two interleaved realizations of F_1 are sufficient to create two tokens in place p_2 to make F_2 realizable. But this is possible only if place p_1 holds initially at least two tokens. If this place holds one token, F_1 is

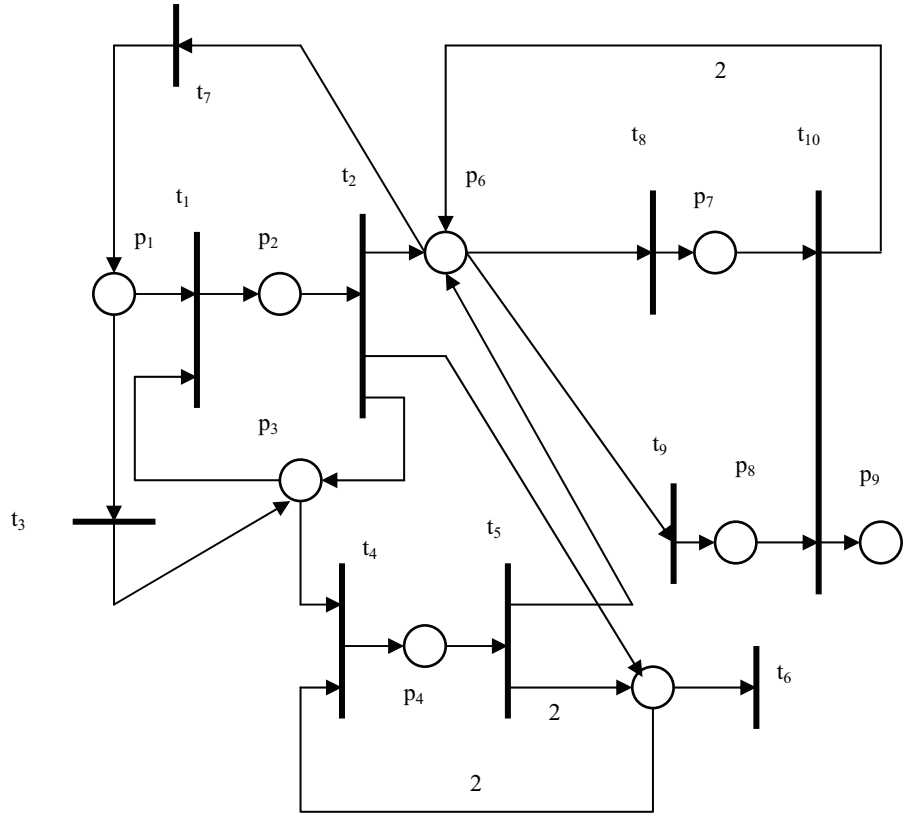
sequentially realizable but it can never create two tokens in p_2 to facilitate the realizability of F_2.

In general, coefficient k_j calculated as was described for the two cases can result in a larger number of realizations of T-invariant F_{nc}^{j} than is actually necessary. The reason is that other T-invariants in (29) can also create tokens in places (36) and contribute to the realizability of F_c.

After computing coefficients k_j in (29), an appropriate method can be applied to find a reachability path (or a legal firing sequence) for the combined T-invariant F if such a path exists. The task here is the following. For a Petri net with given initial and target markings M^0 and M and a combined T-invariant F, find a legal firing transition sequence. To find a legal firing transition sequence, or reachabiity path as defined in (4), known computational techniques can be used (Kostin, 2003; Taoka et al., 2003; Watanabe, 2000; Huang & Murata, 1998).

7. Examples

This section illustrates the proposed reachability analysis scheme by two examples. The examples were tested in Windows XP OS with a prototype C program that implemented



$$D = \begin{bmatrix} -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 & 1 \end{bmatrix}$$

Fig. 2. Petri net of Example 1 and its incidence matrix.

almost all steps of the scheme, with the major exception of the sub-algorithm for solving an ILP problem. To solve this problem, the interactive system QS was used (Chang & Sullivan, 1996). For the first example, Fig. 2 shows a Petri net consisting of $m = 10$ transitions and $n = 9$ places, with its incidence matrix (recall that rows correspond to transitions), and the initial and target markings $M^0 = [2, 0, 0, 0, 0, 0, 0, 0, 0]$ and $M = [2, 0, 0, 0, 0, 0, 0, 0, 1]$, respectively. To get the complemented Petri net, the algorithm appends a row $\Delta M = M^0 - M = [0, 0, 0, 0, 0, 0, 0, 0, -1]$ to the original incidence matrix. Minimal-support T-invariants of the corresponding complemented Petri net are two non-complementary T-invariants $F_1 = [0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0]$ and $F_2 = [1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0]$, and one singular complementary T-invariant $F_3 = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]$, with the sets of affected places $\{p_1, p_3, p_4, p_5, p_6\}$, $\{p_1, p_2, p_3, p_5, p_6\}$ and $\{p_6, p_7, p_8, p_9\}$, respectively. Thus, all these T-invariants are connected and should be considered together. The borrowing matrix G for this example contains the following data:

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
F_1	-1	0	-1	-1	2	-1	0	0	0
F_2	-1	-1	1	0	-1	-1	0	0	0
F_3	0	0	0	0	0	2	-1	-1	-1

Thus, each of these T-invariants can become realizable if it borrows tokens in some of common affected places. Specifically, F_1 needs to borrow two tokens in place p_5, F_2 needs to borrow one token in place p_3, and F_3 borrows two tokens in place p_6. Note that a token borrowed by F_2 in place p_3 can be produced by F_1 in a single realization. In its turn, F_2 is capable, in a single realization, to lend one token to F_1, instead of necessary two tokens. Therefore, F_1 and F_2 can help each other to become realizable. Together, they are capable to produce 2 tokens in place p_6 to be borrowed by F_3.

The desired number of tokens in p_5 can be accumulated if the subnet corresponding to F_2 is not structurally bounded. To check this, the ILP problem (33) for F_2 is solved, in the following form:

$$\text{minimize } a = y_1 + y_2 + y_3 + y_5 + y_6,$$

$$\text{subject to: } -y_1 + y_2 - y_3 \le 0,\ -y_2 + y_3 + y_5 + y_6 \le 0,\ -y_5 \le 0,\ y_1 - y_6 \le 0,\ y_1, y_2, y_3, y_5, y_6 \ge 1.$$

This ILP problem has no feasible solution. Thus, the subnet corresponding to F_2 is not structurally bounded, so that at least one of its affected places is not structurally bounded. We are interested in accumulating two tokens in p_5, so that $\Delta M = [0, 0, 0, 2, 0]$. Therefore, now the ILP problem (37) should be attempted, in the following form:

$$\text{minimize } b = x_1 + x_2 + x_6 + x_7,$$

$$\text{subject to: } -x_1 + x_7 \ge 0,\ x_1 - x_2 \ge 0,\ -x_1 + x_2 \ge 0,\ x_2 - x_6 \ge 2,\ x_2 - x_7 \ge 0,$$

$$x_1 + x_2 + x_6 + x_7 \ge 1.$$

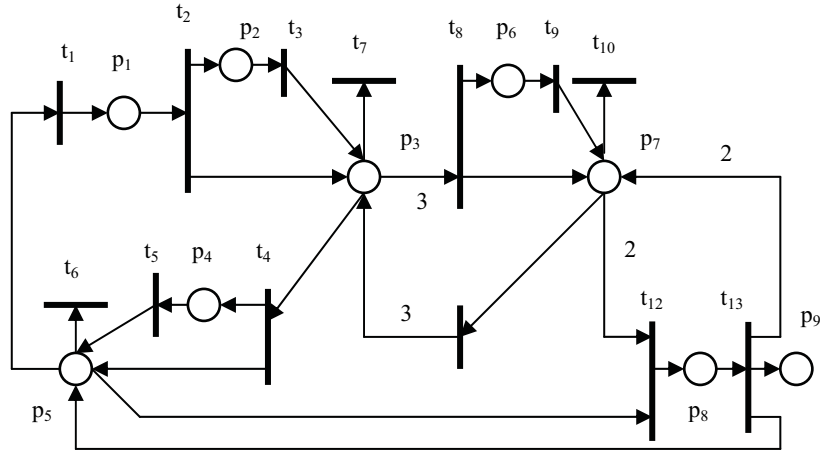
This ILP problem has the optimal (minimal) solution

$$X^* = [x_1^*,\ x_2^*,\ x_6^*,\ x_7^*] = [2,\ 2\ ,2,\ 0].$$

Now, using (38), one can find that

$$k_2 = \max\left(\left\lceil \frac{x_i^*}{f_{2i}} \right\rceil \mid\ i = 1,\ 2,\ 6,\ 7\right) = 2.$$

Since T-invariant F_2 borrows only one token in place p_3 and this token can be created during a single realization of F_1, it is sufficient to have $k_1 = 1$. Thus, the combined T-invariant (29), with $F_c = F_3$, is $F = F_1 + 2F_2 + F_3 = [2, 2, 1, 1, 1, 2, 3, 1, 1, 1, 1]$. For this T-invariant, a legal firing sequence can be found consisting of 15 firing transitions $t_3t_1t_2t_7t_1t_2t_4t_5t_6t_6t_8t_9t_{10}t_7t_7$ and transforming M^0 into M. This is the shortest sequence although there exist other sequences of the same length. Using the computed sequence, the corresponding reachability path (4) from M^0 to M can be easily found.



$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & -1 & 1 \end{bmatrix}.$$

Fig. 3. Petri net of Example 2 and its incidence matrix.

Fig. 3 shows the second example of a Petri net, consisting of m = 13 transitions and n = 9 places. With the initial and target markings M^0 = [1, 0, 0, 0, 0, 0, 0, 0, 0] and M = [1, 0, 0, 0, 0, 0, 0, 0, 1], there are seven minimal-support T-invariants in the corresponding complemented Petri net: six non-complementary T-invariants F_1 = [1, 1, 1, 2, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0], F_2 = [2, 2, 2, 1, 1, 0, 3, 0, 0, 0, 0, 0, 0, 0], F_3 = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0], F_4 = [0, 0, 0, 0, 0, 0, 3, 1, 1, 0, 2, 0, 0, 0], F_5 = [0, 0, 0, 3, 3, 6, 0, 1, 1, 0, 2, 0, 0, 0], F_6 = [2, 2, 2, 1, 1, 0, 0, 1, 1, 2, 0, 0, 0, 0], and one singular complementary T-invariant F_7 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1], with the sets of affected places {p_1, p_2, p_3, p_4, p_5}, {p_1, p_2, p_3, p_4, p_5}, {p_3, p_6, p_7}, {p_3, p_6, p_7}, {p_3, p_4, p_5, p_6, p_7}, {p_1, p_2, p_3, p_4, p_5, p_6, p_7}, and {p_5, p_7, p_8, p_9}, respectively.

Thus, all these T-invariants are connected. Linear combinations of F_7 with non-complementary T-invariants, according to Section 3, yield four additional minimal singular T-invariants F_8 = [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1], F_9 = [2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 1, 1], F_{10} = [0, 0, 0, 2, 2, 4, 1, 1, 1, 0, 2, 1, 1, 1] and F_{11} = [0, 0, 0, 1, 1, 2, 2, 1, 1, 0, 2, 1, 1, 1].

For reachability analysis, consider the singular complementary T-invariant F_7. For F_7 and its connected non-complementary T-invariants, the borrowing matrix G contains the following data:

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
F_1	-1	-1	-2	-1	-2	0	0	0	0
F_2	-1	-1	-2	-1	-2	0	0	0	0
F_3	0	0	3	0	0	-1	-2	0	0
F_4	0	0	6	0	0	-1	-2	0	0
F_5	0	0	6	-1	-2	-1	-2	0	0
F_6	-1	-1	-2	-1	-2	-1	-2	0	0
F_7	0	0	0	0	1	0	2	-1	-1

Thus, T-invariant F_7 can become realizable if only it borrows tokens. Specifically, F_7 needs to borrow one token in place p_5 and two tokens in place p_7. The necessary number of tokens in the both places can be produced by realizable T-invariant F_6 alone. Indeed, F_6 creates, in a single realization, two tokens in place p_5 and two tokens in place p_7. However, at this point we cannot say that there exist a state of the Petri net in which places p_5 and p_7 hold at least one and two tokens, respectively. To learn this possibility, it is necessary initially to test the structural boundedness of the subnet corresponding to F_6, by attempting to solve the ILP problem (33), in the following form:

$$\text{minimize } a = y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7,$$

$$\text{subject to: } y_1 - y_5 \le 0,\ -y_1 + y_2 + y_3 \le 0,\ -y_2 + y_3 \le 0,\ -y_3 + y_4 + y_5 \le 0,\ -y_4 + y_5 \le 0,$$

$$-3y_3 + y_6 + y_7 \le 0,\ -y_6 + y_7 \le 0,\ -y_7 \le 0,\ y_1, y_2, y_3, y_4, y_5, y_6, y_7 \ge 1.$$

This ILP problem has no feasible solution. Thus, the subnet corresponding to F_6 is not structurally bounded, so that at least one of its affected places is not structurally bounded. We are interested in having at least one token in p_5 and at least two tokens in p_7, so that $\Delta M = [0, 0, 0, 0, 1, 0, 2]$. Therefore, now it is necessary to try to solve the ILP problem (37), in the following form:

$$\text{minimize } b = x_1 + x_2 + x_3 + x_4 + x_5 + x_8 + x_9 + x_{10},$$

$$\text{subject to: } x_1 - x_2 \ge 0,\ x_2 - x_3 \ge 0,\ x_2 + x_3 - x_4 - 3x_8 \ge 0,\ x_4 - x_5 \ge 0,\ -x_1 + x_4 + x_5 \ge 1,$$

$$x_8 - x_9 \ge 0,\ x_8 + x_9 - x_{10} \ge 2,\ x_1 + x_2 + x_3 + x_4 + x_5 + x_8 + x_9 + x_{10} \ge 1.$$

This ILP problem has the optimal (minimal) solution

$$X^* = [x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_8^*, x_9^*, x_{10}^*] = [3,\ 3,\ 2,\ 2,\ 2,\ 1,\ 1,\ 0].$$

Now, using (38), we can find that

$$k_6 = \max\left(\left\lceil \frac{x_i^*}{f_{6i}} \right\rceil \mid\ i = 1,\ 2,\ ...,\ 5,\ 8,\ 9,\ 10\right) = 2.$$

Thus, the combined complementary T-invariant (29), with $F_c = F_7$, is $F = 2F_6 + F_7 = [4, 4, 4, 2, 2, 0, 0, 2, 2, 4, 0, 1, 1, 1]$. For F, a legal firing sequence can be found consisting of 26 transition firings and transforming M^0 into M. This is not the shortest sequence. The shortest sequence exists for the decremented value of coefficient $k_6 = 1$ and consists of 14 transition firings $t_2t_3t_4t_1t_2t_3t_5t_8t_9t_{12}t_{13}t_1t_{10}t_{10}$.

Although non-complementary T-invariants F_1 and F_2 are realizable as well, they are not appropriate to be combined with F_7 to create a realizable combined complementary T-invariant since they cannot produce tokens in place p_7. The necessary number of tokens could be produced in p_7 also by F_5 but it needs itself to borrow six tokens in place p_3.

In this way, one can proceed with the remaining singular T-invariants F_8, F_9, F_{10}, and F_{11}. Calculating coefficient $k_6 = 2$ and combining each of these T-invariants with F_6, it will be possible to successfully find the corresponding legal firing sequences and, if necessary, reachability paths. In all cases, coefficient k_6 can be decremented to one, to get the shortest legal firing sequence.

This example shows that, in general, it is not necessary to compute coefficients k_j for all T-invariants F_{nc}^{j} in (29). The reachability test can be done as soon as coefficient k_j is computed for the first F_{nc}^{j}. If this test fails, then coefficient k_j is computed for the next F_{nc}^{j}, until the reachability test is successful or all connected non-complementary T-invariants in (29) are considered.

8. Conclusion

A new approach to reachability analysis in general Petri nets is proposed, formally described, and illustrated by examples tested with a prototype program. For a given original Petri net, the reachability analysis is reduced to the computation and investigation of T-invariants of the complemented Petri net consisting of the original Petri net and an additional, complementary transition with input and output arcs depending on the given initial and target markings. It is shown that, without the loss of reachability information, one can carry out reachability analysis using only a finite number of T-invariants.
We did not address, in this chapter, complexity aspects of the proposed approach to reachability analysis. Complexity of some problems of Petri nets, including the reachability problem, was investigated elsewhere (Jones et al., 1977). Most of the running time in the proposed reachability analysis scheme will be spent in computing minimal-support T-invariants and their linear combinations, solving ILP problems, and trying to find legal firing sequences for the computed T-invariants. This can be done with the use of existing methods (Watanabe, 2000; Yamauchi & Watanabe, 1998; Huang & Murata, 1998).

9. References

Alaiwan, H. & Toudic, J.-M. (1985). Recherche des semi-flots, des verrous et des trappes dans les reseaux de Petri. *Technique et Science Informatique*, Vol. 4, No. 1, 1985, pp. 103 – 112, ISSN 0752-4072.

Bause, F. & Kemper, P. (1994). QPN-Tool for the Qualitative and Quantitative Analysis of Queuing Petri Nets. *Lecture Notes in Computer Science*, Vol. 794, 1994, Springer, Berlin, pp. 321 – 334, ISBN 3540580212.

Boel, R.K.; Ben-Naoum, L. & van Breusegem, V. (1995). On Forbidden State Problems for a Class of Controlled Petri Nets. *IEEE Transactions on Automatic Control*, Vol. 40, No. 10, October 1995, pp. 1717 – 1731, ISSN 0018-9286.

Chang, Y.-L. & Sullivan, R.S. (1996). *QS: Quant System*, Version 2.1, Prentice-Hall, Englewood Cliffs, 1996, ISBN 013239054X.

Caprotti, O.; Ferscha, A. & Hong, H. (1995). *Reachability Test in Petri Nets by Groebner Bases*, Technical Report No. 95-03, Johannes Kepler University, Austria.

Chiola, G.; Franceschinis, G.; Gaeta, R. & Ribaudo, M. (1995). GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, Vol. 24, No. 1&2, 1995, pp. 47 – 68, ISSN 0166-5316.

German, R.; Kelling, C.; Zimmerman, A. & Hommel, G. (1995). TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. *Performance Evaluation*, Vol. 24, No. 1&2, 1995, pp. 69 – 87, ISSN 0166-5316.

Goldberg, J.L. (1992). *Matrix Theory With Applications*, McGraw-Hill Education - Europe, 1992, ISBN 0071129282.

Goodrich, M.T. & Tamassia, R. (2002). *Algorithm Design: Foundations, Analysis and Internet Examples*, John Wiley & Sons, ISBN 0471383651.

Holloway, L.E.; Krogh, B.H. & Giua, A. (1997). A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems*, Vol. 7, No. 2, April 1997, pp. 151 – 190, ISSN 0924-6703.

Howell, J.A. (1971). Exact Solution of Linear Equations Using Residue Arithmetic. *Communications of the ACM*, Vol. 14, No. 3, 1971, pp. 180 – 184, ISSN 0001-0782.

Huang, J.S. & Murata, T. (1998). A Constructive Method for Finding Legal Transition Sequences in Petri Nets. *Journal of Circuits, Systems, and Computers*, Vol. 8, No. 1, 1998, pp. 189 – 222, ISSN 0218-1266.

Jones, N.D.; Landweber, L.H. & Lien, Y.E. (1977). Complexity of Some Problems in Petri Nets. *Theoretical Computer Science*, Vol. 4, 1977, pp. 277 – 299, ISSN 0304-3975.

Kodama, S. & Murata, T. (1988). *On Necessary and Sufficient Reachability Condition for Some Subclasses of Petri Nets*, Technical Report #UIC-EECS 88-8, University of Illinois at Chicago, June 1988.

Kostin, A.E. (1997). The Novel Algorithm for Determining the Reachability in Acyclic Petri Nets. *SIGACT News*, Vol. 28, No. 2, June 1997, pp. 70 – 79, ISSN 0163-5700.

Kostin, A.E. (2003). Reachability Analysis in T-Invariant-less Petri Nets. *IEEE Transactions. on Automatic Control,* Vol. 48, No. 6, 2003, pp. 1019 - 1024, ISSN 0018-9286.

Kostin, A.E. (2006). A Reachability Algorithm for General Petri Nets Based on Transition Invariants. *Lecture Notes in Computer Science*, Vol. 4162, 2006, pp. 608 – 621, Springer, Berlin, ISBN 978-3540377917.

Krukeberg, F. & Jaxy, M. (1987). Mathematical Methods for Calculating Invariants in Petri Nets. *Lecture Notes in Computer Science*, Vol. 266, 1987, Springer, Berlin, ISBN 3540180869.

Kuhn, H.W. & Tucker, A.W. (1956). *Linear Inequalities and Related Systems*. Princeton University Press, 1956, Princeton, NJ.

Mayr, E.W. (1984). An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal of Computing,* Vol. 13, No. 3, 1984, pp. 441 – 459, ISSN 0097-5397.

Memmi, G. & Roucairol, G. (1980). Linear Algebra in Net Theory, *Lecture Notes in Computer Science*, Vol. 84, 1980, pp. 213 – 223, Springer, Berlin, ISBN 978-3540100010.

Murata, T. (1977). State Equation, Controllability, and Maximal Matchings of Petri Nets. *IEEE Transactions on Automatic Control*, Vol. 22, No. 3, June 1977, pp. 412 – 416, ISSN 0018-9286.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol. 77, No. 4, 1989, pp. 541 – 580, ISSN 0018-9219.

Peled, D. (1993). All from One, One from All. *Lecture Notes in Computer Science*, Vol. 697, 1993, pp. 409 – 423, Springer, Berlin, ISBN 978-3540569227.

Peterson, J.L. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981, ISBN 0136619835, Englewood Cliffs, N.J., ISBN 0136619835.

Roch, S. & Starke, P.H. (2001). *INA: Integrated Net Analyzer*, Version 2.2, Humboldt-Universitat zu Berlin, Berlin, 2001.

Schmidt, K. (2000). Stubborn Sets for Model Checking the EF/AG Fragment of CTL. *Fundamenta Informaticae*, Vol. 43, No. 1 – 4, 2000, pp. 331 – 341, ISSN 0169-2968.

Silva, M. & Colom, J.M. (1991). Convex Geometry and Semiflows in P/T Nets. In: Rozenberg, G. (Ed.). *Advances in Petri Nets 1990*, Springer, Berlin, 1991, pp. 79 – 112, ISBN 0387538631.

Springer, J. (1986). Exact Solution of General Integer Systems of Linear Equations. *ACM Transactions. on Mathematical Software*, Vol. 12, No. 1, March 1986, pp. 51 – 61, ISSN 0098-3500.

Takano, K.; Taoka, S.; Yamauchi, M. & Watanabe, T. (2001). Experimental Evaluation of Two Algorithms for Computing Petri Net Invariants. *IEICE Transactions on Fundamentals*, Vol. E84-A, No. 11, 2001, pp. 2871 – 2880, ISSN 1745-1337.

Taoka, S.; Furusato, S. & Watanabe, T. (2003). A Heuristic Algorithm FSDC Based on Avoidance of Deadlock Components in Finding Legal Firing Sequences of Petri Nets. *Lecture Notes in Computer Science*, Vol. 2679, Springer, Berlin, 2003, pp. 417 – 439, ISSN 0302-9743.

Varpaaniemi, K. (1998). *On the Stubborn Set Method in Reduced State Space Generation*, PhD Thesis, Dept. of Computer Science and Engineering, Helsinki University of Technology, Finland, 1998.

Watanabe, T. (2000). The Legal Firing Sequence Problem of Petri Nets," *IEICE Transactions on Information & Systems*, Vol. E83-D, No. 3, March 2000, pp. 397 – 406, ISSN 1745-1361.

Zhou, M. & DiCesare, F. (1993). *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer, 1993, ISBN 0792392897.

20

Reliability Prediction and Sensitivity Analysis of Web Services Composition

Duhang Zhong, Zhichang Qi and Xishan Xu
School of Computer Science, National University of Defense Technology
P.R.China

1. Introduction

Web services are emerging as a major technology for deploying automated interactions between distributed and heterogeneous applications. It aims at the transparent integration of Web applications, based on XML-related standards (F.Curbera et al., 2002). Until now, many research efforts have been made in the field of Web services composition. Moreover, many composition languages have recently emerged, including BPEL, BPML or ebXML, these languages focus on tracking and executing collaborative business processes by business applications.

An important issue for business process built in this way is how to assess the degree of trustworthiness, especially their performance and dependability characteristics. In this paper we focus on reliability aspects, and propose an approach to predict the reliability of web services composition.

Stochastic Petri Nets (SPNs) can be used to specify the problem in a concise fashion and the underlying Markov chain can then be generated automatically. In this paper, we propose the usage of CSPN model, an extension of stochastic Petri nets as a solution to the problems of predicting the reliability of web service composition. The choice of Petri nets was motivated by the following reasons: (a) Petri nets are a graphic notation with formal semantics, (b) the state of a Petri net can be modelled explicitly, (c) the availability of many analysis techniques for Petri nets.

The remainder of this paper is organized as follows. Section 2 provides general information about BPEL and stochastic Petri net. In Section 3 we describe our reliability prediction model and propose an approach to transform BPEL process into CSPN model. Section 4 discusses the result of this mapping on an example BPEL process models. Next, Section 5 discusses the sensitivity analysis of the reliability prediction model. Finally, we discuss the related works and conclude this paper.

2. Background

2.1 BPEL

BPEL, also known as BPEL4WS, build on IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG (Web Services for Business Process Design). It combines the features of a block structured process language (XLANG) with those of a graph-based process language (WSFL). BPEL is intended to describe a business process in two different ways: executable

and abstract processes. An abstract process is a business protocol specifying the message exchange behaviour between different parties without revealing the internal behaviour of any of them. An executable process specifies the execution order between a number of constituent activities, the partners involved, the message exchanged between these partners, and the fault and exception handling mechanisms (Axel Martens, 2005).

A composite service in BPEL is described in terms of a process. Each element in the process is called an activity. BPEL provides two kinds of activities: primitive activities and structured activities. Primitive activities perform simple operations such as receive (waiting for a message from an external partner), reply (reply a message to a partner), invoke (invoke a partner), assign (copying a value from one place to another), throw (generating a fault), terminate (stopping the entire process instance), wait (wait for a certain time), empty (do nothing).

To enable the representation of complex structures, a structured activity is used to define the order on the primitive activities. It can be nested with other structured activities. The set of structured activities includes: sequence (collection of activities to be performed sequentially), flow (specifying one or more activities to be performed concurrently), while (while loop), switch (selects one control path from a set of choices), pick (blocking and waiting for a suitable message). The most important structured activity is a scope. A scope is a means of explicitly packaging activities together such that they can share common fault handling and compensation routines. It consists of a set of optional fault handlers (exceptions can be handled during the execution of its enclosing scope), a single optional compensation handler (inverse some effects which happened during the execution of activities), and the primary activity of the scope which defines its behaviour. (Sebastian Hinz et al., 2005)

The sequence, flow, switch, pick and while constructs provide a means of expressing structured flow dependencies. In addition to these constructs, BPEL provides another construct known as control links which, together with the associated notions of join condition and transition condition, support the definition of precedence, synchronization and conditional dependencies on top of those captured by the structured activity constructs. A control link between activities A and B indicates that B cannot start before A has either completed or has been skipped. Moreover, B can only be executed if its associated join condition evaluates to true, otherwise B is skipped. An activity X propagates a positive value along an outgoing link L if and only if X was executed (as opposed to being skipped) and the transition condition associated to L evaluates to true. Transition conditions are Boolean expressions over the process variables. The process by which positive and negative values are propagated along control links, causing activities to be executed or skipped, is called dead path elimination.

2.2 Stochastic Petri nets

Petri Nets (PNs) is a modeling formalism used for the analysis of a wide range of systems coming from different domains (e.g., distributed computing, telecommunication, control systems, workflow management) and characterized by situations of concurrency, synchronization, causality and conflict (Simona Bernardi, 2003). A PN is basically characterized by places, transitions and weighted arcs defining its structure and it is graphically represented by a directed bipartite graph in which places are drawn as circles, transitions are drawn as bars, input and output arcs are drawn as arrows and inhibitor arcs are drawn as circle headed arrows.

In the original definition of PNs do not include time concepts; temporal specification in PN models was introduced with different approaches, mostly by associating a delay to transitions. In particular, in Stochastic Petri Nets (SPNs) transitions firing delays are exponentially distributed random variables.

Generalized Stochastic Petri Nets (GSPNs) (Marsan A et al., 1995) are an extension of SPNs proposed by M. Molloy in which stochastic timing is mixed with deterministic null delays. In a GSPN model, there are two types of transitions: immediate transitions and timed transitions. Immediate transitions are fired in zero time and used to model logical actions or activities that require a negligible time; while timed transitions are characterized by exponentially distributed firing delays.

Definition 2.1 A GSPN model is a 6-tuple $(P, T, F, W, M_0, \lambda)$, where P is a finite set of places. T is a finite set of transitions partitioned into two subsets: T_I (immediate) and T_D (timed) transitions, where transitions $t \in T_D$ are associated with rate λ. $F \subseteq (P*T) \cup (T*P)$ is a set of arcs. $M_0=\{m_{01}, m_{02}, ..., m_{0k}\}$ is an initial marking. $W : T \to R$ is a function defined on the set of transitions. Timed transitions are associated with priority zero, whereas all other priority levels are reserved for immediate transitions. The immediate transitions are drawn as thin bars, while the timed transitions are drawn as rectangles.

SRNs are an extension of GSPNs (Gianfranco Ciardo et al., 1992), i.e., they include all the features of GSPNs and many more such as guards, timed transition priorities, variable cardinality arcs, halting conditions, and reward rates etc. None of these extensions enhance the modelling power since every SRN model can be converted to a continuous-time Markov chain(CTMC) and CTMCs are isomorphic to GSPNs(although SRNs allow calculation of some reward-based measures which are not possible through GSPNs). Thus any system that can be modelled by a SRN can also be modelled by a GSPN. However, SRNs and GSPNs differ in the conciseness of model specification. SRNs permit a much more concise description of system dependability than GSPNs do.

3. Reliability prediction using CSPN models

3.1 The CSPN model

A basic principle of the SOC paradigms is that each service composition can iteself become a service that can be recursively used in other services' composition. So we distinguish two kinds of service (Vincenzo, 2005):

- *Atomic services* don't require the services of any other resources to perform their tasks. They include, for example, the services offered by basic processing and communication resources but also the services offered by self-contained software components strictly tied to a particular computing platform.
- *A composite service* is realized as a composition of other dynamically selected services that it requires to perform its tasks.

From the reliability prediction viewpoint , the basic difference between these two service types is that the atomic service provider can publish complete reliability information that's directly useful in a service composition's reliability analysis, whereas a composite service provider is only aware of reliability information concerning the part of service implemention under its direct control. The provider must combine this information with the reliability of the other dynamically selected services to get overall service reliability. Hence, to support a service composition's reliability prediction, composite service must provide their service-usage profile, a description of the generated pattern of external service requests

As pointed out by Jens Happe (Jens Happe&Viktoria Firus, 2005), most of the reliability prediction models are based on Markov models. A Markov model can be seen as a finite state machine, whose transitions are annotated with a probability of taking the transition from its source states. These models can be appropriate when dealing with sequential systems. However, as soon as a concurrent or parallel software system (e.g. web service composition) has to be analyzed, different influences come into play, which can hardly be expresses by finite state machines or the corresponding Markov model.

To represent the service-usage profile of web services composition, we propose the Composite Service Process Net model(CSPN) based on the Stochastic Petri Net. In the CSPN model, the basic activity is represented by timed transition, the structure is represented by the immediate transitions and firing rules.

Definition 3.1 A CSPN model is a 4-tuple $\Sigma=(N,\Omega,s,t)$,where:

- N is a GSPN or SRN;
- Ω is the set of external services' operation;
- s represents the starting place of process, a token in the place indicates the service is ready to start.
- t represents the finished place of process, a token in the place indicates the service is terminated.

In the CSPN model, we can distinguish two types of transitions: operation transition represents the invoke of external services; while internal transitions represent the internal activity.

3.2 Transformation of BPEL process into CSPN model

The transformation details of primitive and structured activities into CSPN can be illustrated by these examples in Fig.1. Each primitive activity is represented by one transition. A sequence of activities is represented by the sequential concatenation of one Petri net pattern for each of the activities. A flow activity provides parallel execution and synchronization of activities, two immediate transitions are used to split the control flow into concurrent threads and join them at the end. A switch activity supports conditional routing between activities; the probability of each branch is represented by the weight of immediate transition. BPEL's while activity supports iterative performance of a specified iterative activity. The iterative activity is performed until the given Boolean while conditions no longer holds true. A pick activity exhibits the conditional behaviour where decision making is triggered by external events or system timeout. It has a set of branches in the form of an event associated with it, and exactly one of the branches is selected upon the occurrence of the event associated with it.

Control links are non-structural constructs used to express control dependencies between activities. Each activity within a flow can be source and/or target of several links. Fig 2 depicts the mapping of a linked activity X. The activity X has two incoming and two outcoming links. Each link is transformed into two places lst("link status true") and lsf("link status false") reflecting the Boolean value of that link. Before the activity X can be executed, all incoming links have to be evaluated with respect to the join condition. In Fig 2, the subnet enclosed in the box labelled specifies the mapping of incoming links to activity X, Here the join condition "AND" is defined. In general, each join condition over n links could be expressed by immediate transitions. The subnet enclosed in the box labelled specifies the mapping of outgoing links from activity X, once it is complete; it is ready to evaluate transition conditions to determine the link status for each of the outgoing links.

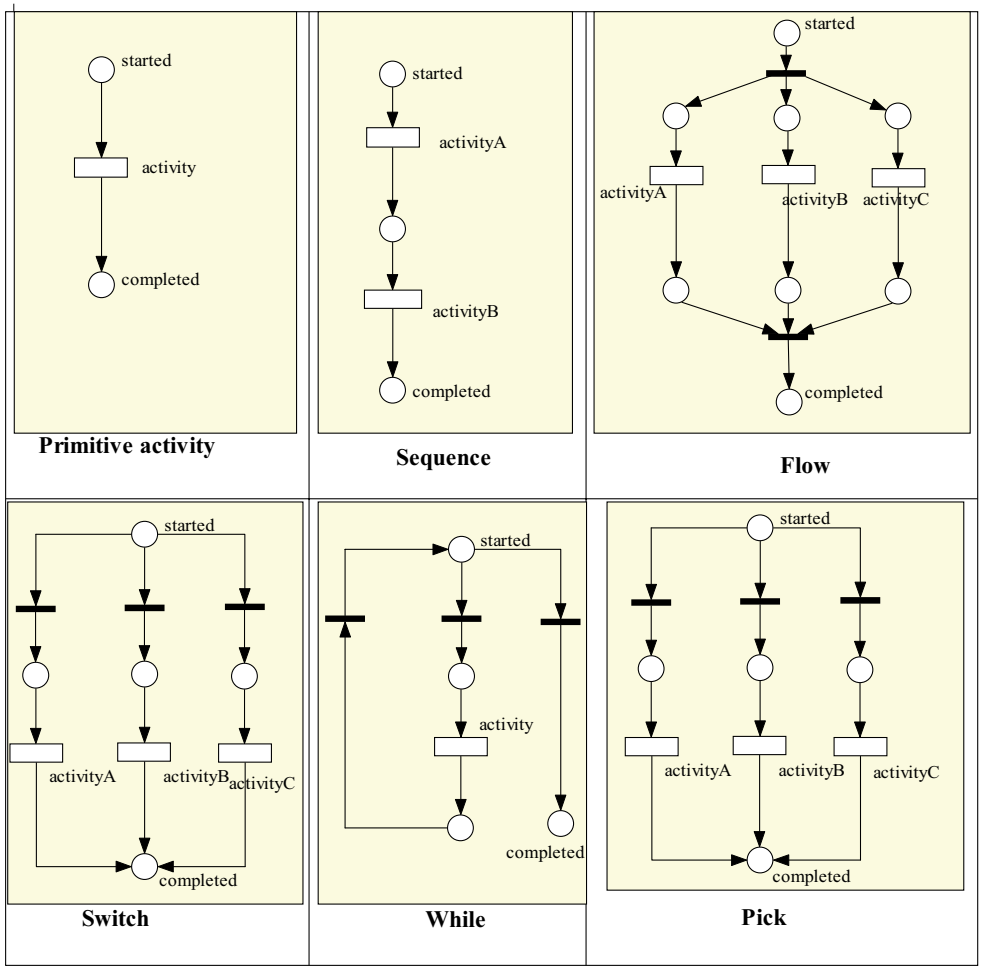


Fig 1. Transformation of BPEL into CSPN

If the join condition evaluates to true, the activity X can start as normal. Otherwise, a fault called join failure occurs. A join failure can be handled in two different ways, as determined by the suppressJoinFailure attribute associated with activity X. If this attribute is set to "yes", the join failure will be suppressed, as modelled by transition "sjf"("suppress join failure"). In this case, the activity will not be activated and the status of all outgoing links will be set to FALSE. If the activity lies on a path of an alternative branch that was not chosen, all outgoing links have to be set to false too. In that case, the activity will not be activated. Instead, it will get a token on the place negLink("propagate negative link values"). The negLink pattern sets all outgoing links to FALSE and propagates the negLink token towards the embedded activities. This is known as dead path elimination.

3.3 Computing the reliability prediction

In the next step, we annotate the CSPN model with the dependability attributes, and derive the reliability prediction of web service composition. There are three kinds of dependability attributes to be annotated:

- For every timed transition which represents the execution of a primitive activity, we annotate the execution time of the activity, which is assumed to be exponentially distributed with mean.
- For every immediate transition which represents the control structure relationship (eg. switch or while), we annotate to describe the weight assigned to the firing of enabled immediate transition t.

In this paper, the reliability measure of a web service we use is the probability of its ability to successfully carry out its own task when it is invoked. To associate the failure behaviour with the activities, we extend the CSPN model transformed from BPEL in section 3.2. For each transition representing the execution of an activity offered by a web service, two immediate transitions added to represent the events that results produced by the activity are correct and incorrect respectively, and have weights (the reliability of the web service) and . This process is depicted as Fig. 3, Place "Fail" represents the failure of the BPEL composite web service.

```
<flow>
  <link name="link1"/>
  <link name="link2"/>
  <link name="link3"/>
  <link name="link4"/>
  ...
  <activityX suppressJoinFailure="yes">
    <targets>
      <joinCondition>
        link1 AND link2
      </joinCondition>
        <target linkName="link1"/>
        <target linkName="link2"/>
    </targets>
    <sources>
      <source linkName=""/>
        <transitionCondition>
          P(...)
        </transitionCondition>
      <source linkName=""/>
        <transitionCondition>
          NOT P(...)
        </transitionCondition>
    </sources>
  <.activityX>
```

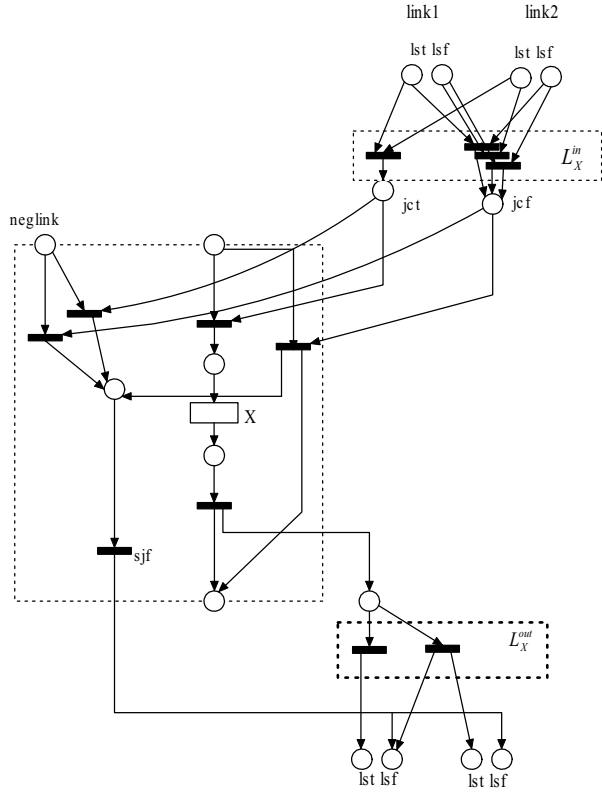


Fig. 2. Transformation of linked activity

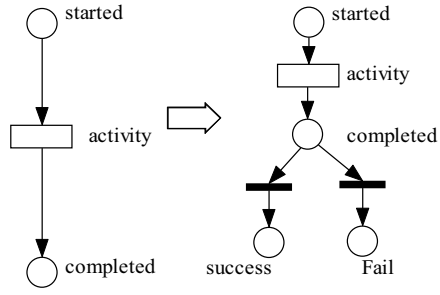


Fig. 3. Associate the failure behavior

The last step is to solve the stochastic Petri net model and compute the reliability prediction of web service composition. In this paper, we use the Stochastic Petri Net Package (SPNP) (C.Hirel et al., 2000) to computation of the reliability measures. SPNP is a versatile modelling tool for stochastic Petri net model; it allows the specification of SPN models, the computation of steady-state, transient, cumulative, time-averaged, and up-to-absorption measures and sensitivities of these measures. The most powerful feature of SPNP is the ability to assign reward rates at the net level and subsequently compute the desired measures of the system being modelled. Here we assign reward rate 1 to all markings in which there is no token in place "Fail"; all other markings are assigned a reward rate equal to zero. And the reliability of BPEL composite web service is the expected reward rate in steady state.

4. Examples

The following example shows how the structure of a BPEL process model is transformed into a stochastic Petri nets model. Fig.4 is the schematic illustration of the example taken from the section on structured activities of the BPEL 1.1 specification (BEA et al., 2003).

This example considers a simple loan approval web service that provides a port where customers can send their requests for loans. Customers of the services send their loan requests, including personal information and amount being requested. Using this information, the loan service runs a simple process that results in either a “loan approved” message or a “loan rejected” message. The approval decision can be reached in two different ways, depending on the amount requested and the risk associated with the requester. For low amounts (less than $10,000) and low-risk individuals, approval is automatic. For high amounts or medium and high-risk individuals, approval is to be studied in greater detail. The corresponding stochastic Petri nets model is depicted as Figure 5.

In this example, the following parameters must be assigned a value before the SRN model can be evaluated:

- the reliability of each partner
- the probability weights of the immediate transitions
- the execution time of each primitive activity

We assume the values given in Table 1. Using the SPNP 6.0, we compute the reliability prediction for the loan approval process as Rel=0.948=94.8%

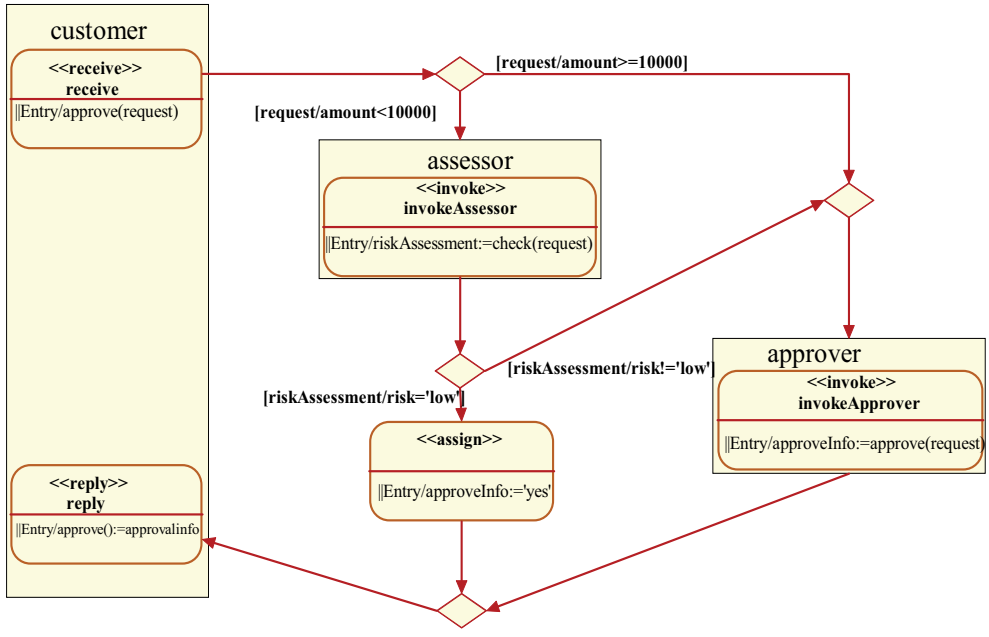


Fig. 4. Loan approval process

Reliability	Value
$R_{Customer}$	0.98
$R_{Assessor}$	0.99
$R_{Approver}$	0.99

Probability	Value
$pr\{amounts <= 10000\}$	0.4
$pr\{amounts > 10000\}$	0.6
$pr\{risk = low\}$	0.3
$pr\{risk = high\}$	0.7

Execution time	Value
$T_{receive}$	4
T_{reply}	4
T_{assign}	1
$T_{invoke\_Assessor}$	10
$T_{invoke\_Approver}$	15

Table 1. The parameters of loan approval process

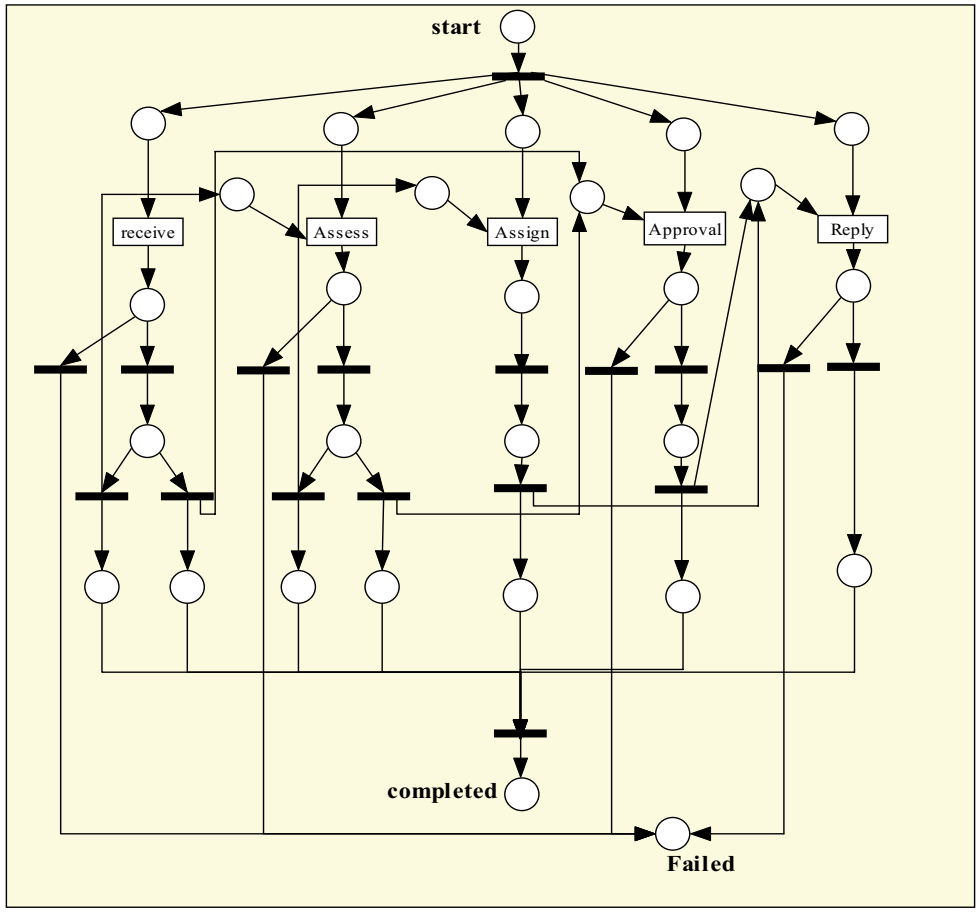


Fig. 5. The SPN model of loanapproval process

5. Sensitivity analysis

In this section, we illustrate some sensitivity analyses that can be performed for our reliability prediction technique: (1) as a function of the component service's reliability and (2) as a function of the usage profile. These analyses are exemplified using the loan approval process example.

5.1 System reliability as a function of component services' reliability

This analysis consists in varying the system reliability as a function of the component services' reliabilities with the purpose of identifying the component service that have the greatest impact on the reliability of the composite service. The method consists of varying the reliability of one component service at a time and fixing the others to 1. The probability distribution of the usage profile is same as section 4. Fig 6 shows the graphs of the reliability

of the composite service as a function of the component services' reliabilities. Note that the component service Approver has a large impact on the composite service's reliability, as the composite service invokes Approver more frequently than Assessor.

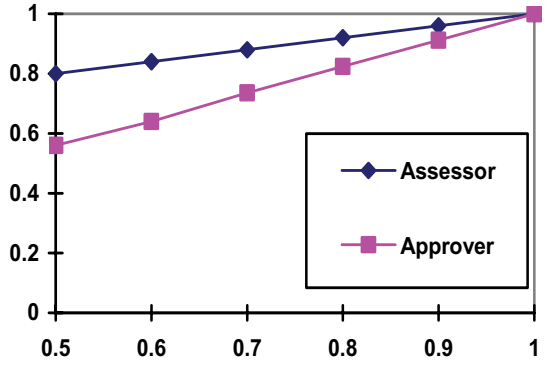


Fig. 6. Reliability as a function of component services' reliabilities

5.2 System reliability as a function of usage profile

This analysis consists in varying the system reliability as a function of the usage profile, as depicted in the graphs of Fig 7. The reliabilities of component services are same as section 4. If we vary the probabilities of low amounts and low risk from 0.3 to 0.9, the result is shown in Fig 7. Note that the usage profile does not have much impact on the system reliability, as the reliabilities between the component service Approver and Assessor are very close in this simple example.

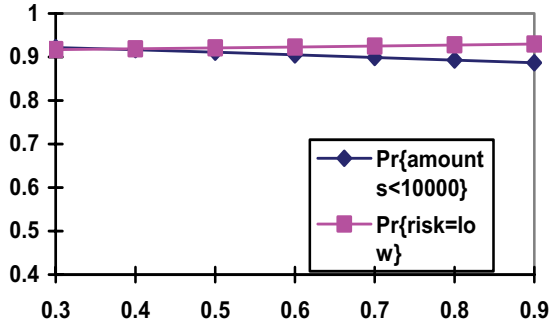


Fig. 7. Reliability as a function of service-usage profile

6. Related works

Approaches to the reliability analysis of service- and component-based system have been already presented. According to the classification proposed by Goseva Popstojanova (Goseva Popstojanova, 2001), they can be divided into two main categories: state-based approaches and path-based approaches. For the sake of brevity, we provide here a brief view of the approaches of greatest interest to the scope of this work.

State-based models (R.C.Cheung, 1980) use a control flow graph to represent the system architecture. In such models it is assumed that the transfer of control among the components

can be modelled as a Markov chain, with further behaviour of the system dependent only on the current state. The architecture of software has been modelled as a discrete time Markov Chain (DTMC), continuous time Markov Chain (CTMC), or a semi-Markov process (SMP). These can be further classified into absorbing and irreducible. The former represents applications that operate on demand which software runs that correspond to terminating execution can be clearly identified. The latter is well suited for continuously operating software applications, such that in real time control systems, where it is either difficult to determine what constitutes a run or there maybe very large number of such runs if it is assumed that each cycle consists a run.

Path-based models (S.M.Yacoub et al., 1999) compute the reliability of the system by enumerating possible execution paths of the program. The model used in their approach is the component dependency graph (CDG), this reliability analysis technique is specific for component based software whose analysis is strictly based on execution scenarios. A scenario is a set of component interactions triggered by specific input stimulus, and it is related to the concept of operations and run-types used in operational profiles (D.Musa, 1993).

Vincenzo Grassi present an approach to the reliability prediction of an assembly of services, that allows to take into account in an explicit and composition way the reliability characteristics of both the resources and interaction infrastructures used in the assembly (Vincenzo Grassi, 2005). What distinguishes their approach is the exploitation of a "unified " service model that helps in modelling and analyzing different architectural alternatives, where the characteristic of both "high level" services and "low level" services are explicitly taken into consideration. Moreover, this work also point out the importance of considering the impact on reliability of service sharing.

Apostolos focused on the development of a principled methodology for the dependability analysis of composite Web Services (Apostolos Zarras et al., 2004). The first step involves a UML representation for the architecture specification of composite web services. The proposed representation is built upon BPEL and introduces necessary extensions to support the dependability analysis. The automated mapping of this extended UML models to traditional dependability analysis models such as Block Diagrams, Fault Trees and Markov models is the core of the methodology.

7. Conclusion

In this paper, we introduce an approach to predict the reliability of Web services composition. We present the transformation algorithms from BPEL, which is the de facto industry standard of Web services composition specification, to CSPN models. Using the model, we can compute the reliability prediction of the web service composition. The major contribution of this paper is a reliability prediction technique that takes into account the structure of BPEL specification and the concurrent nature of service composition. For future work, we will transform all control-flow constructs of BPEL (including link, scope, faultHandler etc) into Petri nets. And we will use our CSPN model to give a more precise estimation of the reliability and performance of web service composition.

8. References

Axel Martens (2005), Analyzing Web Service based Business Processes. *In Proc. of FASE'05*, Edinburgh, Scotland.

Apostolos Zarras, Panos Vassiliadis, and Valerie Issarny(2004), Model-Driven Dependability Analysis of Web Services, *In Proceedings of the International Conference on Distributed Objects and Applications (DOA)*,LNCS3291.

BEA, IBM, Microsoft, SAP AG, and Siebel Systems (2003), Business process execution language for web services (version 1.1). ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

C. Hirel, B. Tuffin, and K. S. Trivedi (2000), SPNP: Stochastic Petri Nets. Version 6.0, in Computer performance evaluation: Modelling tools and techniques, *11th International Conference; TOOLS 2000, Schaumburg, Il*., USA, B. Haverkort, H. Bohnenkamp, C. Smith(eds.), LNCS 1786.

F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana(2002), Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI, *IEEE Internet Computing*, 6(2). pp86–93.

Gianfranco Ciardo, Jogesh K. Muppala and Krishor S. Trivedi (1992), "Analyzing Concurrent and Fault-Tolerant Software using Stochastic Reward Nets", *Journal of Parallel and Distributed Computing, Vol. 15*, pp. 255-269.

H.M.W. Verbeek and W.M.P. van der Aalst(2005), Analyzing BPEL Processes using Petri Nets, *In Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management,* Florida International University, Miami, Florida, USA, pp.59-78.

JD.Musa (1993), Opeartional profiles in software reliability engineering, In *IEEE Software 10(2).*

Jens Happe, Viktoria Firus (2005), Using Stochastic Petri Nets to Predict Quality of Services Attributes of Component-Based Software Architectures, *the Tenth International Workshop on Component-Oriented Programming*, Glasgow, Scotland.

K.Goseva-Popstojanova, A.P. Mathur, K.S.Trivedi(2001), Comparison of architecture-based software reliability models, *In Proc. Of the 12th Int. Symposium on Software Reliability Engineering (ISSRE 2001).*

Marsan A, Balbo G, Conte G, Donatelli S, Franceschinis G (1995), Modelling with Generalized Stochastic Petri Nets, Wiley, Chichester, England.

R.C.Cheung (1980), A User-Oriented Software Reliability Model, *In IEEE) Transactions on Software Engineering, volume 6(2)*, PP 118-125.

R.H. Reussner, H.W.Schmidit, I.H.Poernomo (2003), Reliability prediction for component-based software architectures., *Journal of Systems and Software,* no.66,pp241-252.

Sebastian Hinz, Karsten Schmidt, and Christian Stahl (2005), Transforming BPEL to Petri Nets, In *Proc. 3rd Int. Conf. on Business Process Management (BPM 2005)*, LNCS 3649, Nancy, France, pp. 220-235.

S.M.Yacoub,B.Cubic, and H.H.Ammar (1999), Scenario-Based Reliability Analysis of Component-Based Software, *In Proc. of the 10th ISSRE*, Boca Raton, FL, USA.

Simona Bernardi (2003), Building Stochastic Petri Net models for the verification of complex software systems, PHD Paper, Torino.

Vincenzo Grassi(2005), Architecture-Based reliability Prediction for Service-Oriented Computing, *Architecting Dependable Systems III*, LNCS 3549,pp.279-299.

W-L,Wang, Y.Wu, M-H Chen(1999), An Architecture-based software reliability model, *Proc. IEEE Pacific Rim Int. Symposium on Dependable Computing*, Hong Kong China.

Zhangxi Tan, Chuang Lin, Hao Yin, Ye Hong, and Guangxi Zhu(2004), Approximate Performance Analysis of web Services Flow Using Stochastic Petri Net, *In GCC 2004*, LNCS 3251,pp.193-200.

21

Petri Nets for Component-Based Software Systems Development

Leandro Dias da Silva[1], Kyller Gorgônio [2] and Angelo Perkusich[2]
[1]Paraiba State University, [2]Federal University of Campina Grande
Brazil

1. Introduction

The Software Engineering discipline was created to try to apply techniques and methods of others engineering disciplines to software systems development. To achieve this goal it was necessary to change the way software was developed, not only at code level, but also at the process level. Like in other engineering disciplines, one of the major objectives of software engineering is to develop artifacts in a systematic way. Several building block approaches were proposed and developed along the years. Nowadays one of the most researched and used approach are software components (Crnkovic and Grunske, 2007. Nierstrasz et al., 2002). Components are autonomous units with independent life cycle that represent an specific functionality. A component consists of functionality, interface and possibly other non functional characteristics.

The development of bigger systems with components as building blocks is called Component Based Development (CBD). To make this possible it is necessary to adapt the traditional software engineering techniques and methods, or even defined new ones, to attend to specific CBD requirements. In the context of Component Based Software Engineering (CBSE) the objective is to define a set of practices that promotes the CBD.

Formal methods improve the development process of software and hardware systems by helping designers to achieve dependability at different levels of abstractions such as requirements, specification, modeling and design. This is mainly due to the fact that the application of formal methods helps discovering and removing errors by performing automatic analysis and verification (Clarke and Wing, 1996). Petri nets (Murata, 1989), and more specifically Hierarchical Coloured Petri Nets (HCPN) (Jensen, 1992. Jensen, 1997) are a very powerful tool that has been widely studied and applied for the specification and analysis of complex concurrent systems (Donatelli and Thiagarajan, 2006. Kleijn and Yakovlev, 2007. Jensen, 2005. Jensen, 2006). It has a graphical representation that helps the design of complex software systems. There are several advantages of using a formal method in systems design such as, automatic simulation, proof of properties and unambiguous documentation.

In the context of software engineering, the reuse of artifacts in the development of new software systems increases the productivity. Also, the reuse of artifacts that are well known to be correct is an effective way to increase the dependability on the system under development. Reuse is not restricted to pieces of source code, but it can be also be applied to

requirements, documentation, project decisions and specifications. During the last few years, component based software engineering has been applied to promote the development of software systems based on reuse (Szyperski, 1999. Crnkovic , 2001. Crnkovic et al., 2002. Crnkovic and Grunske, 2007. Nierstrasz et al., 1992. van Steen et al., 1998).

The basic premise during the reuse process is that the designer should observe that in specific application domains, different software systems share some common characteristics. These characteristics can be represented by any kind of artifact, such as source code or a model described using some formal language, HCPN in the context of this chapter. Therefore, the identification of such common characteristics is a very important task. Firstly, when an artifact that has been already modeled is identified, it is possible to search for it in a repository and reuse it with some adaptations, instead of modeling it from scratch. Secondly, after an artifact is modeled and verified, it can be made available to be reused in the future.

The main goal of this chapter is to introduce a systematic and automatic approach to the reuse of HCPN models in the specification and verification of complex software systems. The focus is on the study and development of techniques that help the automation of the modeling phase, reducing time and money costs of the project. This approach is an alternative to *ad-hoc* reuse practices in which the reuse process is of the entire responsibility of the developer. In order to achieve this goal the approach for the specification and analysis of components, frameworks for components composition, and component-based software systems is presented. The proposed approach is guided by a reuse process and software tools for automatic manipulation of the models. Moreover, a case study is used to illustrate its application. The work presented in this chapter is based on the use of temporal logic (Emerson, 1990), model checking (Clarke et al., 1999. McMillan, 1993) and supervisory control theory (Ramadge & Wonham, 1989) in order to support: adaptation, integration and use verification of HCPN models. It is fully implemented in CPN/ML language (Christensen & Haagh, 1996) for a well known set of tools for HCPN models called Design/CPN (Des, 2006). From the application point of view, the introduced approach is used to develop models in the context of complex embedded software systems. Embedded systems have been applied in several kinds of computing devices (Nierstrasz et al., 2002) such as automobiles, cellular phones and control and automation devices. Due to the evolution of the technology, more complex devices executing more complex tasks are being developed, making it difficult to deal with the increasing complexity from the software point of view (Lee, 1999. Lee, 2002). As discussed by Knight (Knight, 2001. Knight, 2002. Knight, 2004), two major problems that must be tackled in this domain are specification and verification. The first one is mainly related to the need to build models that are more dependable. The later one is related to the difficulty in performing tests on embedded software systems. Therefore techniques such as model checking can help to early detect design errors. The approach introduced in this chapter is an effective approach to deal with these two problems.

The rest of the chapter is organized as follows. In section 2 the basics of the formal tools used in the present work are introduced, including a discussion about Coloured Petri nets, temporal logic and model checking. The application of reuse techniques to build formal models is discussed in Section 3. In Section 4 an example of an embedded system to illustrate the reuse process is presented. The adaptation, integration, and use verification

steps are described in Sections 5, 6, and 7, respectively. Finally, in Section 8 the chapter is concluded with suggestions for future work.

2. Preliminaries

2.1 Petri nets

Petri nets are a formal method with strong mathematical foundation and a graphical representation. The mathematical foundation promotes the use of automatic analysis and verification techniques. On the other hand the graphical notation avoids the use of possibly ambiguous textual notations or hard to understand mathematical notations. Petri nets can be used in the design of complex systems, expressing properties such as precedence relationships, conflicts, concurrency, synchronization, deadlocks, and resource sharing among others. Also, the state and action locality characteristic allow the modeling of complex systems using either bottom-up or top-down approaches. Therefore, it promotes modularity and reusability that are important characteristics for the modeling solution presented in this work.

As mentioned in the introduction, in the context of this chapter an extension of Petri nets called Hierarchical Coloured Petri Nets (HCPN) is used as a description language. This extension incorporates complex data types and hierarchy concepts to Petri nets. An HCPN is a set of non-hierarchical CPN models, and each CPN model is called a CPN page. Therefore, an HCPN is an extension of the concept of CPN that allows the modeling in hierarchical levels. This is possible due to the inclusion of two mechanisms: substitution transition and fusion places. A substitution transition is a transition that represents a CPN page. The page in which the substitution transition belongs to is called *superpage* and the page represented by that transition is called *subpage.* The association between subpages and superpages is done by means of sockets and ports. Sockets are all the input and output places of the substitution transition in the superpage. Ports are the places in the subpage associated to the sockets. The ports can be of input, output, or input-output type. For simulation and state space calculation, sockets and ports are glued together and the resulting model is a flat CPN model.

The fusion places are physically different but logically only one, defined by means of a fusion set. Therefore, all the places belonging to a fusion set have always the same marking.

A marking of a place is the set of tokens in that place in a given moment. And the marking of a net is the set of markings of all places in the net, in a given moment. When a marking of a place belonging to a fusion set changes, the marking of all places belonging to that set also changes.

Indeed, these two additional mechanisms, substitution transition and fusion places, are only a graphical notation that promotes the organization and visualization of a CPN model more efficiently. They favor the modeling of larger and more complex systems because the designer can obtain a model by either abstraction or composition, or even both. In order to manipulate tokens in a CPN, it is defined the concept of multi-set, that is, a set where it is possible to have several occurrences of the same element. This concept allows similar parts of the model to be modeled as token information instead of structure replication.

In the following, the definition of CPN according to (Jensen, 1992) is presented.

Definition 1: A Coloured Petri net is a tuple CPN = (Σ, P, T, A, N, C, G, E, I) satisfying the requirements below:

1. Σ is a finite set of non-empty types, called colour sets.

2. P is a finite set of places,
3. T is a finite set of transitions,
4. A is a finite set of arcs such that:

$$P \cap T = P \cap A = T \cap A = 0$$

5. N is a node function defined from A into (P x T) U (T x P).
6. C is a colour function defined from P into S.
7. G is a guard function defined from T into expressions such that:

$$\forall t \in T: [Type(G(t)) = B \wedge Type\left(Var(G(t))\right) \subseteq \Sigma]$$

8. E is an arc expression function defined from A into expressions such that:

$$\forall a \in A: [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$$

9. I is an initialization function defined from P into closed expressions such that:

$$\forall p \in P: [Type(I(p)) = C(p)_{MS}]$$

The definition of HCPN according to (Jensen, 1992) is presented as follows.
Definition 2: A Hierarchical CPN is a tuple HCPN = (S, SN, SA, PN, PT, PA, FS, FT, PP) satisfying the following requirements:

1. S is a finite set of pages such that:
   - Each page s ∈ S is a CPN:
     $(\Sigma_S, P_S, T_S, A_S, N_S, C_S, G_S, E_S, I_S)$
   - The set of net elements are pair wise disjoint:
     $\forall s_1, s_2 \in S: [s_1 \neq s_2 \rightarrow (P_{s_1} \cup T_{s_1} \cup A_{s_1}) \cap (P_{s_2} \cup T_{s_2} \cup A_{s_2}) = \emptyset$
2. SN ⊆ T is a set of substitution nodes,
3. A is a page assignment function defined from SN into S such that:
   - No page is subpage of itself:
     $\{s_0 s_1 \cdots s_n \in S^* | n \in N_+ \wedge s_0 = s_n \wedge \forall k \in 1..n: s_k \in SA(SN_{s_{k-1}})\} = \emptyset$
4. PN ⊆ P is a set of port nodes.
5. PT is a port type function defined from PN into {in, out, i/o, general},
6. PA is a port assignment function defined from SN into binary relations such that:
   - Socket nodes are related to port nodes:
     $\forall t \in SN: PA(t) \subseteq X(t) \times PN_{SA(t)}$
   - Socket nodes are of the correct type:
     $\forall t \in SN, \forall (p_1, p_2) \in PA(t): [PT(p_2) \neq general \implies ST(p_1, t) = PT(p_2)]$
   - Related nodes have identical colour sets and equivalent initialization expressions:
     $\forall t \in SN, \forall (p_1, p_2) \in PA(t): [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$
7. FS ⊆ P_s is a finite set of fusion sets such that:
   - Members of a fusion set have identical colour sets and equivalent initialization expressions:
     $\forall fs \in FS, \forall p_1, p_2 \in fs: [C(p_1) = C(p_2) \wedge I(p_1) <>= I(p_2) <>]$
8. FT is a fusion type function defined from fusion sets into {global, page, instance} such that:
   - page and instance fusion sets belong to a single page:

$$\forall fs \in FS[FT(fs) \neq global \Rightarrow \exists s \in S: fs \subseteq Ps]$$

9. PP $\in S_{ms}$ is a multi-set of prime pages.

Based on Definitions 1 and 2 several activities are defined to manipulate nets using the Design/CPN tool in order to develop and maintain a reuse-based modeling process for complex software system, as detailed in the following sections. Detailed explanations for these definitions, as well as the dynamic behavior of CPN and HCPN are omitted. For more information the reader can see the references (Jensen, 1992. Jensen, 1997).

2.2 Temporal logic

Temporal logic is a modal logic that can be used to describe how events occur over the time. There are operators to describe safety, liveness and precedence properties, providing a framework to specify software systems, particularly concurrent systems (Pnueli, 1977). Temporal logics are used to predicate over the behavior of a system defined by a Kripke structure. This behavior is obtained starting from an initial state and then repeatedly moving from one state to another following the transition relation. It means that such relation should be total, and as consequence all the behaviors of the system are infinite. Since a state can have more than one successor, the structure can be thought of as unwinding into an infinite tree, representing all the possible executions of the system starting from its initial states.

Two useful temporal logics are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). They differ in how they handle branching in the underlying computation tree. The CTL operators permit to quantify over the paths departing from a given state. In LTL, operators are intended to describe properties of all possible computation paths. It is an agreement that the temporal logic provides a good framework to describe and to reason about the behavior of concurrent systems. However, it is not the case when the question is which one is more appropriate, linear or branching time logic, to do it. But this is a question that is outside of the scope of this chapter. Along this chapter, we use a Computation Tree Logic (CTL) (Clarke et al., 1999) defined for Coloured Petri Nets named ASK-CTL (Christensen and Haagh, 1996). In what follows the basic concepts of both logics are introduced.

The CTL temporal logic combines path quantifiers with linear time temporal logic operators. The path quantifiers A *("for all paths")* and E *("for some paths")* should be used as a prefix of one of the operators G *("globally"),* F *("sometimes"),* X *("nexttime")* and U *("until").* Let *AP* be set of atomic propositions, then the syntax of CTL is given by the following rules:

1) If $\varphi \in AP$, then φ is a formula, where AP is a set of atomic propositions;
2) If φ_1 and φ_2 are formulae, then $\neg\varphi_1$, $(\varphi_1 \vee \varphi_2)$ and $(\varphi_1 \wedge \varphi_2)$ are also formulae;
3) If φ_1 and φ_2 are formulae, then $EX\varphi_1$, $EG\varphi_1$ and $E[\varphi_1 U \varphi_2]$ are also formulae.

The others CTL operators are expressed using the three operators EX, EG and E[U]. Therefore:

$$AX\varphi \equiv \neg EX\neg\varphi$$

$$EF\varphi \equiv E[\text{true } U\ \varphi]$$

$$AG\varphi \equiv \neg EF\neg\varphi$$

$$AF\varphi \equiv \neg EG\neg\varphi$$

$$A[\varphi_1 U \varphi_2] \equiv \neg E[\neg\varphi_2\ U\ (\neg\varphi_1 \wedge \neg\varphi_2)] \wedge \neg EG\neg\varphi_2$$

The semantic of CTL is defined with respect to paths in a Kripke structure. A path is an infinite sequence of states (s_0, s_1,...) such that s_{i+1} is reached from s_i for all $i \geq 0$. So, if φ is a CTL formula M, $s \vDash \phi$ is used to denote that φ holds for s_0 of M.
The four most used CTL operators are *EF, AF, EG*, and *AG*. In Fig. 1 the interpretation for such operators is illustrated in an intuitive way, and they are interpreted as follows.

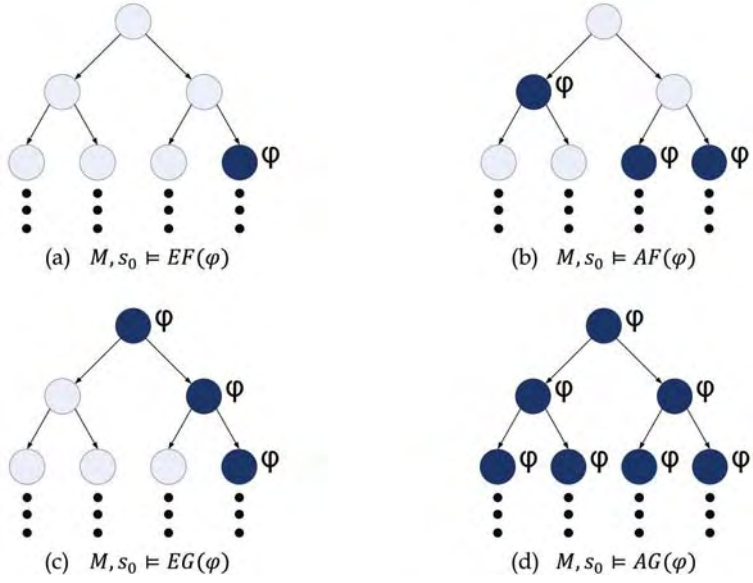


Fig. 1. Basic CTL operators.

EFφ ≡ E[true U φ] means that exists a path starting from s_0 in which φ holds at some state along this path.

AFφ ≡ A[true U φ] means that for all paths starting from s_0, φ holds at some state along the path. In other words, φ is inevitable.

EGφ ≡ ¬AF¬φ means that exists a path starting from s_0 in which φ holds at every state along this path.

AGφ ≡ ¬EF¬φ means that for all paths starting from s_0, φ holds at every state along that paths. In other words, φ holds globally.

2.3 ASK-CTL

ASK-CTL is a CTL-like logic useful to specify properties for CPNs (Coloured Petri Nets) state spaces, represented by occurrence graphs. Occurrence graphs carry information on both nodes and edges. Hence, a natural extension for CTL is to include the possibility to

express properties about the information labeling for the edges (e.g., edge information is needed when expressing liveness properties since liveness is expressed by means of transition occurrence information). For this purpose two mutually recursively defined syntactic categories of formulae are defined: state and transition formulae, which are interpreted over the state space for states and transitions respectively (Cheng et al., 1997). As in CTL, quantified state formulae and transition formulae are interpreted over paths. Path quantification is used in combination with the *until* operator to express temporal properties.

The ASK-CTL library has two parts: one which implements the ASK-CTL logic language and another one which implements the model checker (Christensen and Haagh, 1996). The syntax of ASK-CTL is minimal and in order to increase the readability of the formulae we make use of syntactic sugar, e.g., POS(φ) means that it is possible to reach a state where φ holds, INV(φ) means that φ holds in every reachable state, and EV(φ) means that for all paths φ holds within a finite number of steps.

2.4 Model Checking

The need to increase the dependability of software systems motivates the definition and application of more dependable developing methods and techniques. This need is more evident when dealing with critical real-time systems. With the increasing complexity of the systems the traditional methods based on tests, for example, are not enough anymore to guarantee dependability.

The use of formal methods can increase the confidence in the behavior of the system. In the specification, formal methods can be used to find difficult errors before developing the real system. Traditional methods based on tests and simulation can detect initial errors. But after the simplest errors are fixed, more rigorous methods are needed.

Model checking is used to verify specifications (Clarke et al., 1999) in an exhaustive way. That is, where tests and simulations analyze some possibilities, formal methods analyze all possible behaviors.

One great advantage of model checking is that it is fully automatic. Moreover, the model checking algorithms generate a counter-example generation in case of negation of a property indicating a path where the property is false.

The disadvantage of model checking is the state explosion problem. That occurs when the system has several concurrent components, or when it manipulates complex data types. Some techniques have been researched and developed to deal with this problem such as symbolic model checking (McMillan, 1993) and partial order reduction (Peled, 1994. Valmari, 1991).

The verification activity consists in checking if a property is satisfied by a model or not. The properties are described in temporal logic, and the models can be described as a finite automaton or as a Petri net, for instance. Let M be a model and f be a temporal logic formula that express some property of M. The model checking consists in verify if M models f, which is noted by $M \models f$.

The model checking consists of the following three activities:

Modeling: The modeling consists in describing the system in some formalism. The formalism to be used depends on the tool to be used in the verification, the designer knowledge, or the culture in the institution that is developing the project. It is still possible to transform a given formalism into another to perform the verification.

Specification: The specification is usually done in temporal logic that is used to specify how a system's behavior evolves over time. It is not possible to guarantee the completeness of the specification, that is, it is not possible to guarantee that all the properties to be verified

are specified. But once a property is specified it can be checked against the model for all its possible behaviors.
Verification: Given a model and a specification the verification is fully automatic. In the case of a property is negated the designer must analyze the counter-example to solve possible modeling errors, or to reformulate the specification. Moreover, abstraction and modular techniques depend on the designer to allow that the verification can be performed dealing with the state explosion problem.

3. Reuse based software modeling

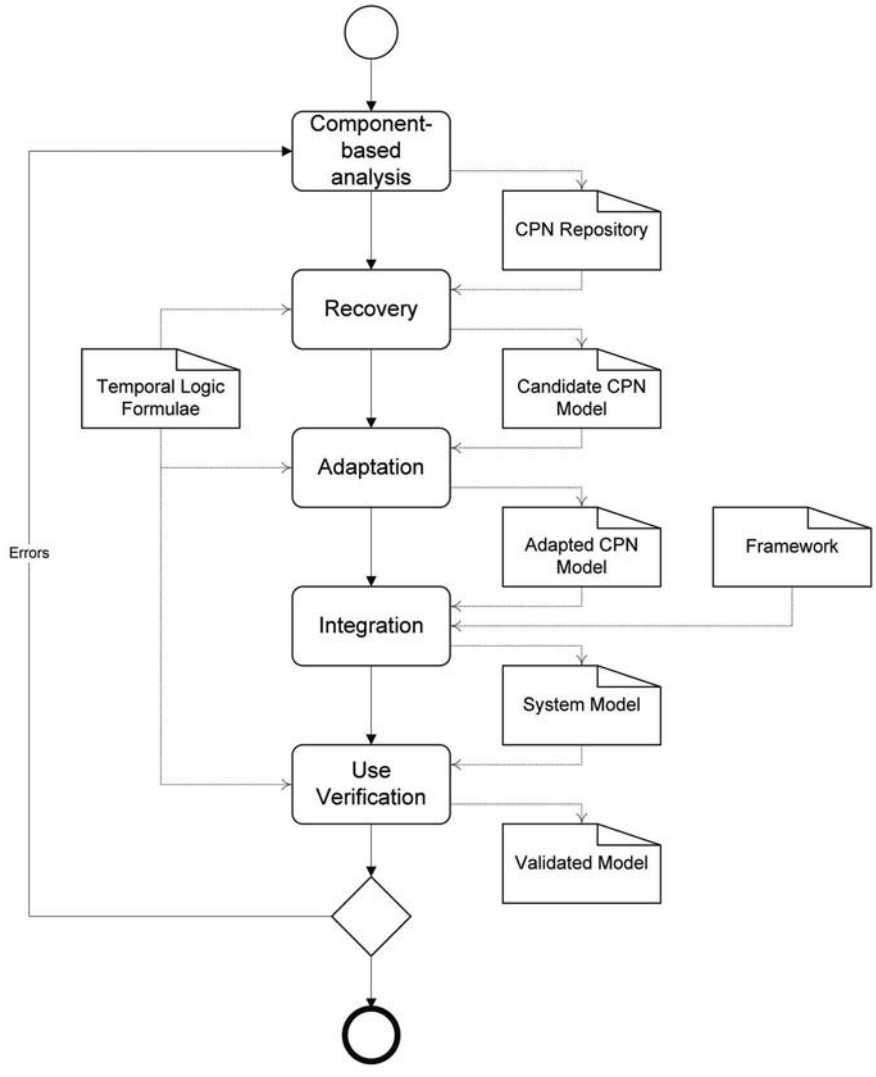


Fig. 2. Diagram for the systematic reuse solution.

When using a reuse based modeling method it is not always necessary to build the hole system from the scratch, it is possible that some of the required models of the system are already modeled. The reuse process defined in this chapter is illustrated by Fig. 2. The main reuse activities are recovery, adaptation, integration and use verification. A detailed discussion of the repository management activity, recovery and insertion of models, can be found in (Lemos & Perkusich, 2001). In this work the adaptation of a recovered model (Gorgdnio & Perkusich, 2002) and the integration of such model into an architectural framework are detailed. The functionality of the reuse process as a whole, unifying all the activities in a systematic modeling method is also discussed.
Besides the reuse activities, a use verification step is taken into account. This step consists in performing model checking in the integrated models in order to verify whether the specific use case is correct, that is, to verify if they were correctly used.
As pointed out in the introduction, an embedded system is used to illustrate the process as well as to guide the definition of the activities.
The designer must think on how and where to search for models that can be directly reused, and adapted if necessary, while building a new system. Moreover, the designer must try to identify potential candidates for reuse and store them in a repository of models. The following reuse activities are identified during the formal modeling of systems:

- Identification of the parts of the new model;
- Definition of a framework;
- Detection of the parts that need to be constructed and those that can be reused;
- Description and recovery of the models that can be reused;
- Adaptation of the recovered models;
- Integration of the recovered/adapted/constructed models;
- Identification of new reusable models and storing them in the repository

It is important to point out that, besides the fact that this technique is fully supported by a set of tools the methodology itself is not completely automatic. The designer plays an important role and is required to create the framework on which the recovered and adapted models are integrated. Moreover, she is required to write down a set of temporal logic formulae describing the behavior of the models to be recovered and adapted.

4. Case study: A component-based embedded system

The application domain considered in the scope of this work is an embedded transducer network control system (Silva & Perkusich, 2005). As shown in Fig. 3, this system is composed by a set of transducers, a controller, named the communication server, and a real-time server.
The environment signals acquired by the sensors are transformed and controlled in a way that the real-time server can access and modify the information to control the actuators according to the application requirements. Observe that the transducers are connected to a controller and that besides control functions it also acts as a front end communication server. Therefore, different applications can be specified and verified by changing the components. Several different applications may access the real-time server to acquire data and to control devices. For instance, it is possible to have temperature, ventilation and humidity sensors. The signals that are acquired and processed can be used to control an HVAC (Heating, Ventilation and Air Conditioning) system in an intelligent building.

It is important to note that a system defined as shown in Fig. 2 is very common in many other kinds of command and control systems and therefore it is possible to define a software architecture that can be reused in other applications.

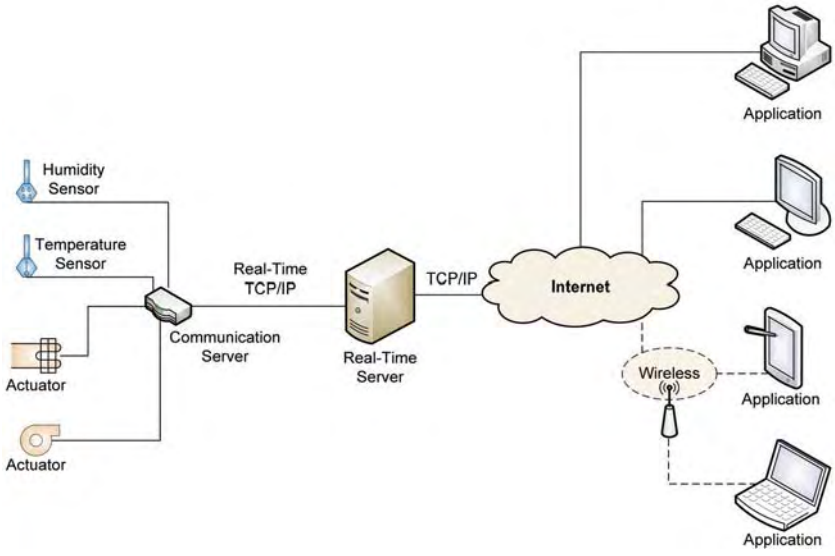


Fig. 3. System structure.

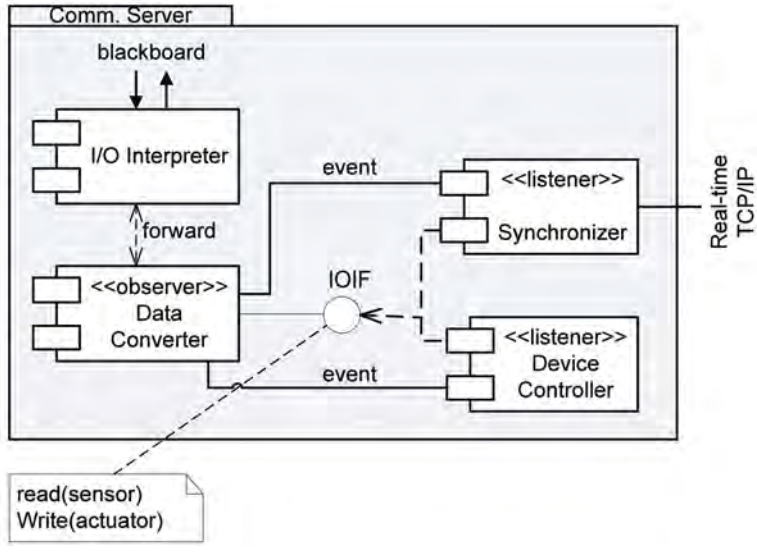


Fig. 4. Communication server

According to the requirements of the applications, defined based on the command and control problem, and the transducers used, different systems can be built. Based on the

approach introduced in this work a system does not need to be specified and verified always from the scratch. What is necessary to do is to recover a model from a repository, modify it to satisfy the new requirements and integrate it on the new project. Only in the case that no model can be found, the designer must specify a new one. To promote this approach, a product line to this specific domain using some specific software architecture to reuse common components is defined. The main advantages are time and money savings, and the reduction or even elimination of errors, and therefore, faults can be avoided. Also, it is possible to maintain and evolve a repository of reusable components for a given domain improving the dependability on the models.

An important observation s that the details related to specific technologies to implement the components are abstracted. The focus is on the specification and analysis of the architecture of a target system. Therefore, properties for the interfaces and architectural level of the components are verified regardless internal details of them. For instance, the protocol used by the communication server to communicate with the control system running in the real-time server is abstracted.

In Figs. 4 and *5 is* illustrated the component diagrams for the communication server and the real-time server, respectively. The communication server consists of four components. The IO Interpreter instantiates raw data from sensors to objects. The Data Converter transforms the data to an specific format. Device Controller is used to calibration, initialization and other control tasks. The Synchronizer is the communication channel with the real-time server. The real-time server is composed by three components. The data controller is used to control data flow among several applications accessing the server. The access to the server is available through the UI component. The real-time server Synchronizer is the counterpart to the communication server Synchronizer.

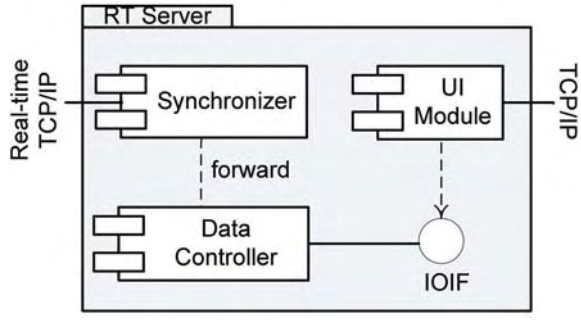


Fig. 5. Real-time server

4.1 Framework

In Fig. 6 the HCPN framework that specifies the architecture of the system is illustrated. There, it is possible to identify how the entities communicate with each other. The System page models the sensors and actuators. They communicate only with the communication server represented by the CommServer page.

There are several components defined for the communication server page. Data from the devices to the embedded system and output data to devices are communicated using a blackboard mechanism. The input and output interpreter, lOlnterpreter, is used to instantiate the data written in the blackboard as objects. Also, this component receives objects from the system and translates them to the data format used by the devices. This component is fixed in the architecture. That is, it is not necessary to change it from one project to another.

The next component is the data converter, DataConvert. This component transforms data from the I/O interpreter to a format used by the real-time server, according to the requirements of the applications. Since data formats are dependent of the applications that access the server this component must be changed to satisfy the requirements of each project. The data converter decides the data flow. If data in the data converter is a control requisition, such as an initialization or calibration request, that data is sent to the device controller. If data is an information signal it is sent to the synchronizer, EmbChannel, to be transmitted to the realtime server.
The device controller component, DeviceControl, is used to control devices, that is, as said before to perform calibration and initialization tasks. Moreover an application can request changes in the attributes for a device, such as, the sampling time. This is done also by the device controller. The synchronizer is a realization of the communication between the communication server and the real-time server. When a sensor sends an information signal and not a control signal, it must be transmitted to the real-time server through the synchronizer. Thus, there is a synchronizer for the communication server and another one for the real-time server. Since this communication does not change, the synchronizer is fixed in the architecture.

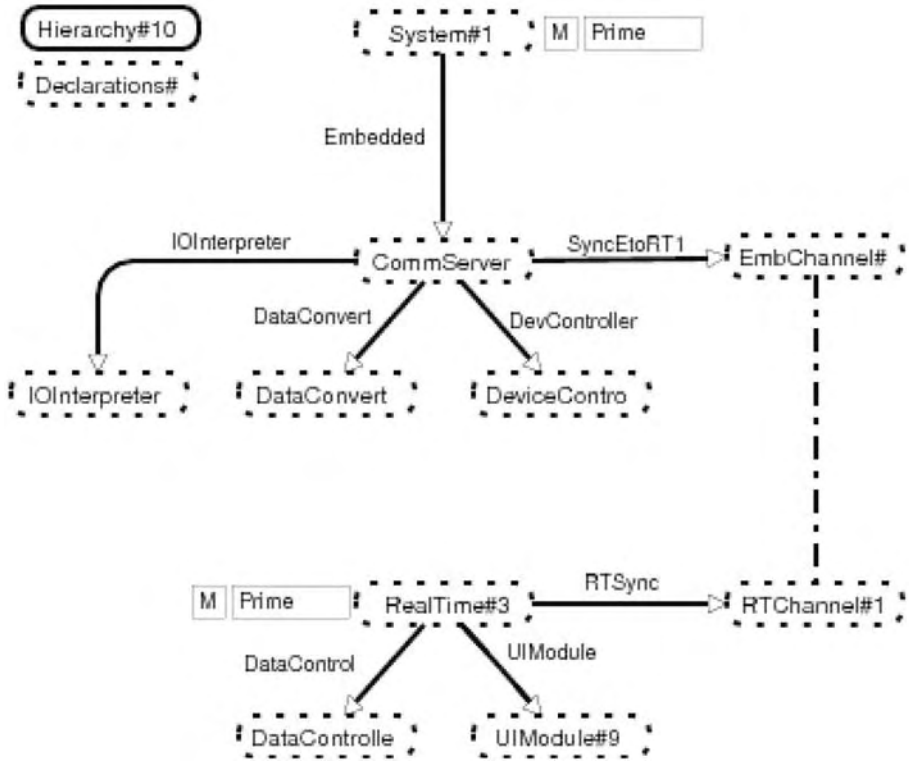


Fig. 6. Model hierarchy (framework).

The real-time server, RealTime, intermediates the communication between the communication server and the applications. A database with information about the net and the applications is used to promote this communication. The applications can read or write

information to control the system. In the real-time server we have several components also. The synchronizer, RTCHannel, is identical to the one in the embedded system. The data controller, DataContoller, is used to control data flow from and to the applications. The user interface module component, UIModule, is used to make services available. The applications use this component to access the system.

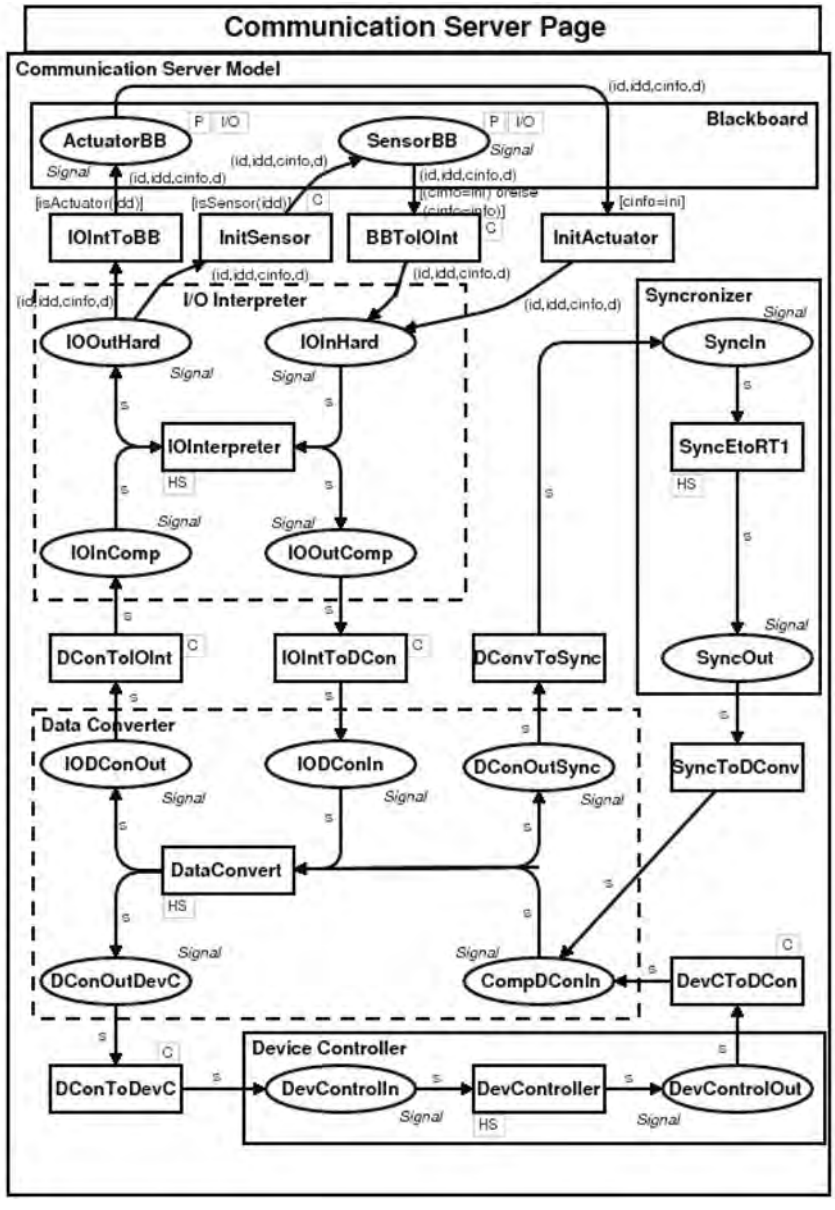


Fig. 7. Embedded system.

In Fig. 7 the dashed lines define components that must be replaced, or hot spots, based on the application, and the continuous lines define components that do not need to be changed or frozen spots. It is possible to see all the components for the communication server. The top part is the blackboard where messages are exchanged between the sensors and actuators and the server. The first component is the I/O interpreter. After this component the signal is sent to the Data Converter and at this point it can take two different destinations, the Device Controller, or the Synchronizer.
Using this architecture, it is possible to specify any control system as defined here, promoting a product line evolution based on the reuse of component models. Moreover, this strategy allows the practice of refactoring at a model level.
The specification described in this section is a general explanation of the model. This model was constructed using the reuse process described in Section 3.

5. Model adaptation

Once a model, that is a candidate to be reused, is identified and recovered from the repository, it is necessary to verify if it is ready to be integrated on the system framework. Usually it is necessary to adapt the recovered model to satisfy some special conditions that holds in the new system. The adaptation technique presented here is based on the use of temporal logic, model checking and supervisory control theory. The basic idea is that for a given CPN model that satisfies some properties, it should be possible to refine it in order to obtain a new model whose behavior is a refinement of the behavior of the original one. Note that on the context of this work, adaptation is a refinement relation. Basically, it means that all the possible behaviors of the adapted model are also allowed in the original model, and in some sense, the models can be related trough a preorder relation (Long, 1993).
In (Ramadge and Wonham, 1987) an algorithm to obtain the supremal controllable sub-language for a given language is described. They assumed that a system, described as a finite automaton, is composed of some events that can be controlled and others that cannot. If the occurrence of a controllable event leads to an undesired situation, it is possible to disable it. However, if the event is not controllable, then it is not possible to do it. For example, in the case of the environmental controller the changes in the temperature of the room, i.e. the data received by the sensors, are not an event that can be controlled by the system. And it makes no sense to change the behavior of the model by avoiding the occurrence of an event that cannot be controlled.
The supremal controllable sub-language algorithm receives as input two finite automata. One modeling the system, m_1 and other modeling the desired behavior of the system, m_2, and the set of events is divided into controllable and uncontrollable. The algorithm returns an automata m_3 that is the maximal, with respect to the behavior allowed by m_2, automata that can be controlled without reaching any undesirable state. In general terms, the algorithm works by removing the undesired states from m_1 until a fix point is reached.
The problem with this direct approach is that it is not possible to know in advance if such m_3 exists or not. It may be the case that it is not possible to refine m_1 until the resulting model satisfies the properties specified by m_2. This is known only after the execution of the algorithm when it returns an empty model. Since it is executed over the state space of the models, if such state space is too large, it is possible that the entire process takes a long time to be executed without generating any useful output.

Model checking techniques can be used to avoid this problem. The idea is to model the automata m_2 as set of temporal logic formulae, more specifically CTL, and use them to check if m_1 can be refined to satisfy the CTL properties or not. Note that it is necessary to determine if there is a subset of m_1 that satisfies the properties. If the model checker outputs a positive result, the synthesis procedure is executed.

Observe that the models to be adapted are given as CPN models. So, the first step in the adaptation procedure is the generation of its state space. Then, it is possible to verify and refine it as described above.

After the execution of the supremal controllable sub-language algorithm, an occurrence graph that is isomorphic to the original one is obtained. The only difference is that the states that should not be reached are marked as undesirables. They are not removed from the occurrence graph of the Petri net.

Next it is necessary to modify the input CPN model in such a way that the state graph of the new model will be isomorphic, considering the label of each arc, to a sub-graph of the original state graph of the input model. Once it is done, the new CPN model is generated.

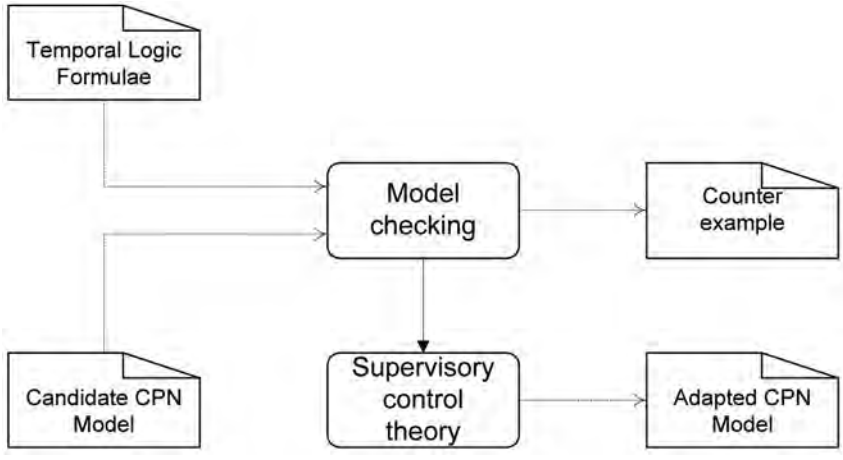


Fig. 8. Adaptation of reusable models.

The adaptation technique introduced here is illustrated in Fig. 8 and the steps required to perform the adaptation are:

1. Generate the occurrence graph of the CPN model;
2. Verify if the CPN model may satisfy the new specification by applying model checking techniques;
   - In the negative case, ask for human intervention and terminate;
3. Execute the supremal controllable sub-language algorithm;
4. Adapt CPN model to generate the new occurrence graph.

Adapting the CPN model consists in adding some control information on it in such a way that the states marked as undesirables are not reached in the new model. In the Design/CPN, each state of the occurrence graph of a model has a unique label. Taking this into consideration, control can be added to the CPN model by creating a new place, called control place, which should be input and output place of all transitions of the CPN model. The initial marking of this control place will be the label of the initial state of the adapted

occurrence graph. Every time a transition occurs, it removes the token on the control place and puts a new one with the label of the new reached state.
The value of the new state is determined by a function that is constructed from the adapted occurrence graph. This function receives as input the value of the token in the control place that represents a state s in the adapted graph and the label t of the transition being executed. The output is the label of the state s' in the adapted graph that is reached from s through the occurrence of t. This control function should be attached to all transitions in the model, i.e. every time a transition occurs the function is executed and the value of the token in the control place is updated.
Finally, it is necessary to add guards to some transitions of the CPN model in order to disable them if its execution in the current state leads to an undesirable state. Note that a transition may be enabled or disabled depending on the value of the marking of the control place. Therefore, if $s_i \xrightarrow{t} s_j$ belongs to the adapted state graph, s_j is marked as an undesirable state, and t models a controlled event defined for the system, a guard is added to transition t to disable it whenever the marking of the control place is the label of s_i. Observe that if t models an uncontrollable event, no guard can be added to it. However, t should always be connect to the control place due to the fact that even if t cannot be controllable, it should be observable.

5.1 Implementation

Algorithm 1: Model Adaptation.

```
1.  OCCGRAPH := MakeStateSpace(CPN);
2.  SCCGRAPH := MakeSCCGraph(OCCGRAPH);
3.  if ASKCTL(properties) == true then
4.      NEW_OCCGRAPH := RamadgeWonham(OCCGRAPH);
5.      CP := CreateControlPlace(CPN);
6.      SetInitialmarking(CP);
7.      CF := CreateControlFunction(NEW_OCCGRAPH);
8.      AddToGolbalDeclarationNode(CF);
9.      for all Transition t in CPN do
10.         AddNewArc(CP, t)
11.         AddNewArc(t, CP);
12.         AddCodeSegmentToTransition(t, CF);
13.     end for
14.     for all State M in NEW_OCCGRAPH not marked as undesirablr do
15.         for all M' such that M -t-> M' do
16.             if M' is marked as undesirable and t is controllable then
17.                 AddGuardToTransition(t, label(M));
18.             end if
19.         end for
20.     end for
21. end if
```

Before presenting the algorithm, the adaptation problem can be stated as follows:
"*Given a CPN model, called CPN, and a set of behavioral restrictions described as CTL formulae, the adaptation problem consists in the synthesis of a new CPN model, called* CPN_{adp}*, that satisfies these new restrictions taking CPN as the starting point.*"

The adaptation strategy described above is defined by Algorithm 1, which is implemented in CPN/ML (Christensen and Haagh, 1996. Ullman, 1998), and it is executed inside the Design/CPN tool as a loadable module.

Lines 1 and 2 define the computation of the occurrence graph (OCCGRAPH) and the strongly connected components graph (SCCGRAPH) of the CPN model. Lines 3 and 4 define the invocations of the ASKCTL model checker and to the implementation of the Ramadge and Wonham algorithm respectively. The other functions are defined to manipulate the internal structures of the Design/CPN models to add the objects used to add control to the model. In Fig. *9* it is illustrated how the adaptation can be performed for the specification of a system.

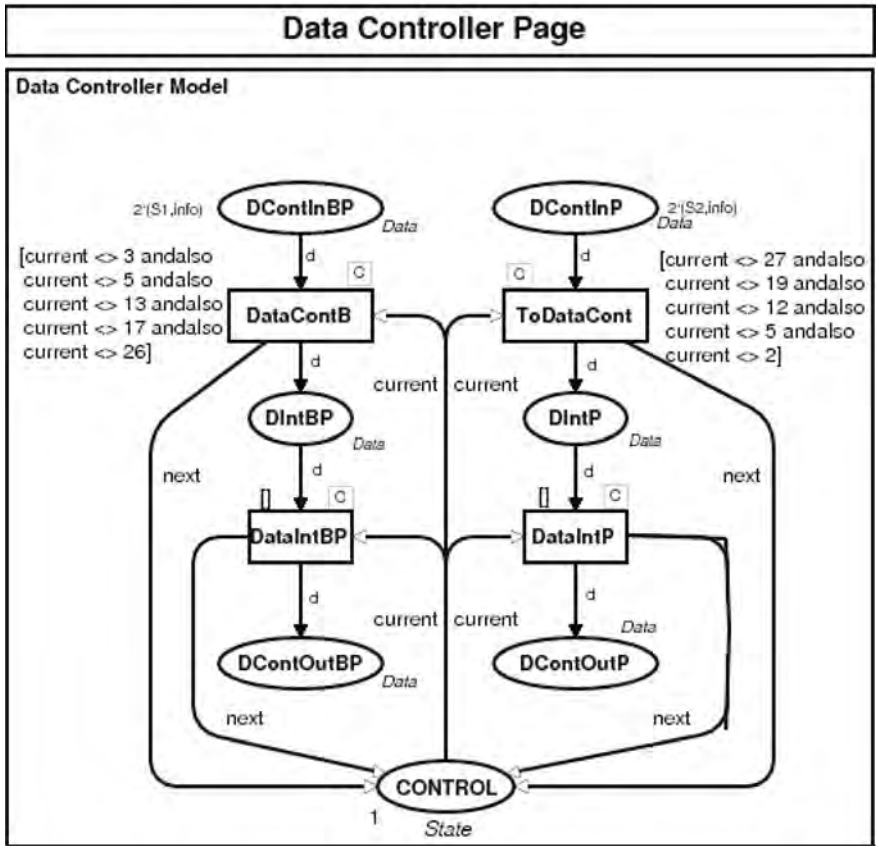


Fig. *9*. Adapted CPN model.

For this example, the component DataController (see framework on Fig. 2) is adapted in order to limit the capacity of the output buffers, DIntBP and DIntP, in such a way that each of them never store more than one token at a given time. This property is expressed in the ASKCTL language by formula:

POS(NOT(MoreThanOne));

Such formula captures the notion that there is a path in the OCC graph for which the evaluation of the function MoreThanOne is false. Such function receives as input a node of the OCC graph and checks if the number of tokens in the places DIntBP and DlntP are greater than one. MoreThanOne is also written in CPN/ML, and it makes explicit references to the elements of the component DataController. This means the designer does not see the component as a black box, and she should have some acknowledgment of the model candidate to be reused.
The code bellow is part of the control function generated by the adaptation procedure. This function takes the label of the current state of the CPN model as input and returns the label of the next state. This information is used to decide whether a transition can be allowed to fire or not.

```
var current: STATE;
var next: STATE;
fun NextState(trans, mark)=
    case (trans, mark) of
        ("DataController'DataContB",  1) => 3 |
        ("DataController'ToDataCont", 1) => 2 |
        ("DataController'DataContB",  3) => 6 |
        ("DataController'ToDataCont", 3) => 5 |
        ("DataController'DataContB",  5) => 8 |
        ("DataController'ToDataCont", 5) => 7 |
        ("DataController'DataContB",  2) => 5 |
        ("DataController'ToDataCont", 2) => 4 |
        ...
```

And finally the guards of transitions DataContB and ToDataCont are added in order to disable them when necessary. Observe that there are no guards on transitions DContOutBP and DContOutP. This is due to the fact that there no situation in which their occurrence leads to an undesirable state, i.e. they cannot increase the amount of tokens in places DlntBP and DlntP.

6. Model integration

After a model is recovered from the repository, and possibly adapted, it has to be integrated into the framework. The integration, as well as the other activities, are fully implemented using the CPN/ML language (Christensen and Haagh, 1996) for the Design/CPN tool set.
First, the designer is asked the name of the file with the CPN model to be integrated into the framework. Then, some functions are automatically executed to build the integration environment, that is, the places, transitions, arcs, and its respective names, color sets, and inscriptions. The next step executed by the algorithm is the definition of the input and output ports in the diagram being integrated. After this step, the substitution transition is defined, and the sockets in the superpage are associated to its respective ports, previously defined in the subpage. The last step is to select the box with the model declarations, in the model page, to define the color set of the ports based on the sockets colors, and to append this information in the global declaration node.
The file selection needs the user interaction, while all other steps are fully automatic executed. To define the algorithm some restrictions are considered. They are:

- Unique page name;
- Prefix in the color sets names indicates the page name;
- Suffix in the place names indicates whether it is a port or not;
- Dot-dashed line patter for the auxiliary box with model declarations;
- Model declarations box must to be unique;
- Declarations of the port places must be the first ones in the model declarations box.

The first restriction to be considered ensures that the page name of the integrated model is unique, and it is the responsibility of the designer. The model being integrated must have a prefix in its color set names with the page name. Another restriction is about port places. The places that are ports must have a suffix IN or OUT, to input and output ports, respectively, in its names. This restriction is to allow the algorithm to recognize which places are ports and which type of port they are.

The last integration restriction is that in the model being integrated there must exist an auxiliary box with dot-dashed line pattern. This box must contain all the declarations for this page. The declaration of the ports color set must be the first ones in this box. This is necessary to the algorithm be able to adapt correctly the color set to successfully integrate the model.

It is important to note that it is the responsibility of the designer to ensure that the restrictions are satisfied in order to the algorithm works properly.

6.1 Implementation

Before presenting the algorithm, the integration problem can be stated as follows:

"*Given a CPN model called CPN, and a Framework called CPNFramework, the integration problem consists in creating places, transitions, arcs, and their respective inscriptions, as well as the hierarchy and append the declarations of CPN to the global declarations of CPNFramework to have a new integrated model taking CPN and CPNFramework as the starting point.*"

The integration is implemented as defined by the Algorithm 2. Steps 2 to 7 are fully automatic. Step 1 needs the user interaction to select the file name of the model to be integrated.

Algorithm 2: Model Integration.

```
1. CPNFramework := CreatePlaces(CPNFramework);
   CreateTrasitions(CPNFramework); CreateArcs(CPNFramework);
2. CPN := DefinePorts(CPN);
3. CPNFramework := DefineSubstitutionTransition(CPNFramework);
4. CPNFramework := AssociatePortsToSockets(CPN, CPNFramework);
5. CPN.PortsColours := CPNFramework.GetSocketsColours();
6. CPNFramework.GlobalDeclaration := CPNFramework.AppendGlobalDeclaration(CPN.GetLocalDeclarations);
```

The integration depends on the framework. Therefore, for each application domain it is necessary to define the architecture of the system and to model it as a CPN framework. For each domain and framework, it is necessary to implement specific functions for the integration step of the reuse process. But this implementation has to be done just one time, and it is used throughout the evolution of the product line in the specific domain with the framework.

Now, it is illustrated how the integration phase of the process is performed. In Fig. 10 it is shown an example where the DataController is integrated into the CommunicationServer model. In this example it is possible to see part of the integration phase, because some parts are internal to the algorithm and to the global declaration node. It is possible to see, for example, how the sockets in the superpage are associated to the ports in the subpage, and how a substitution transition in the superpage represents another CPN model, the page of the component model.

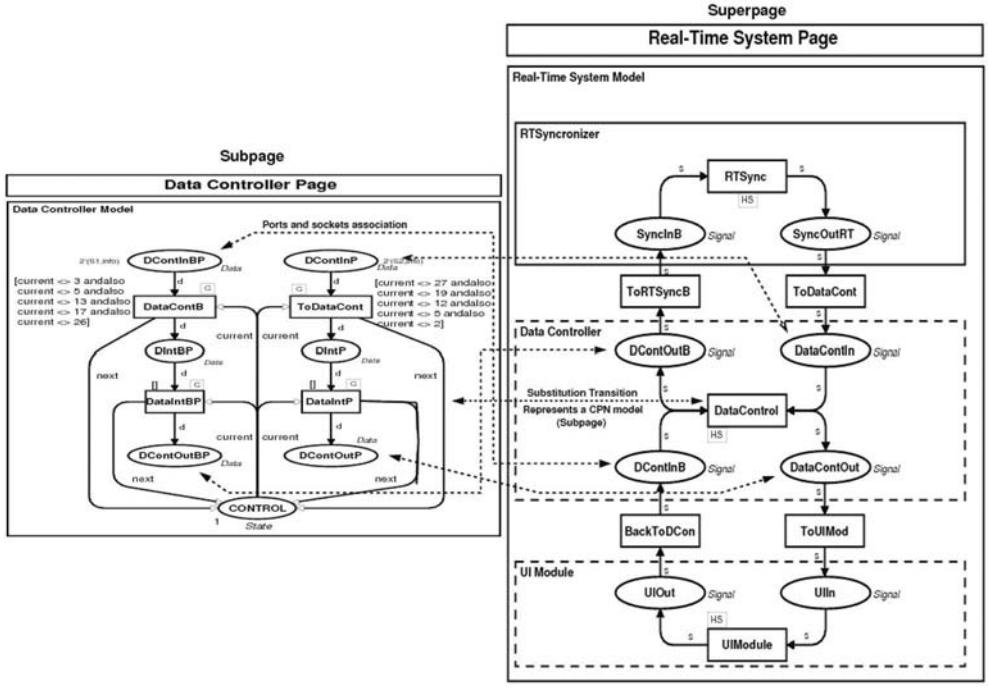


Fig. 10. Integration example.

7. Use verification

Besides adaptation and integration, the specific use case of recovered models performing an use verification step is also considered. This activity is defined in the context of this work because when modeling based on reuse it is necessary to guarantee that the semantic of the resulting model respects the semantics of the reused models. Some parts of the resulting model can lead a reused model to behave in a different way than expected. This problem can compromise the modeling activity and the facility, and flexibility that the reuse process promotes.

The use case verification activity consists in performing model checking for the framework with the individual models to be verified already integrated on it. To do this, the temporal logic formulae for the properties of an individual model is specified in a file,. The model checking is then performed in the whole model to ensure that the integrated models were correctly used. The idea is to use the same specifications used in the recovering step, or the specification supplied together with the model.

Considering that it is necessary to define a new framework for each modeling domain. The framework can be developed without any considerations, or even violations, to the specific functionalities of the individual models to be integrated following the reuse process. Moreover, the models can be reused in several different domains. The interface between the framework and the models can be changed to reflect the needs of each domain and to satisfy the specific use alternatives of each model. Therefore, as the framework is being built the changes in the interface can result in a wrong use case of the integrated models. Because of the problems discussed above, it is necessary to define the use verification activity for the reuse process. Indeed, the use verification activity is essential when building models from a new framework.
Another justification for the definition of the use verification activity is that in the case that no reuse candidate is found in the repository, a new model has to be built. The use verification is also performed to validate a new model to be inserted in the repository.

7.1 Implementation

Before presenting the algorithm, the use verification problem can be stated as follows:
"*Given a CPN model, called CPNFramework, and some properties of an individual component model integrated on CPNFramework, described as temporal logic formulae, the use verification problem consists in performing model checking to verify if the new integrated model respects the individual properties of the integrated component taking CPNFramework and the properties as the starting point.*"
In Algorithm 3 the use verification is defined. Initially, the designer must execute the occurrence graph tool in the Design/CPN. The next step is to select the file that has the properties specifications for the model to be verified. The algorithm executes at this point the model checking in the framework including all the models already integrated on it. If the properties hold in the resulting model the use verification is successful. In the opposed case the designer receives a warning that there exist errors in this specific use case.
The steps 1 to 3 must be done by the user. All other steps are automatically executed. After the execution, a message is shown saying whether the properties are satisfied or not.

Algorithm 3: Use Verification.

```
1.  DELETEOCCGRAPH;
2.  OCCGRAPH := MakeStateSpace(CPNFramework);
3.  SCCGRAPH := MakeSCCGraph(OCCGRAPH);
4.  if ASKCTL(properties) == true then
5.      ShowMessage(SUCCESS);
6.  else
7.      ShowMessage(FAILURE);
8.  end if
```

When the SCC graph is generated and the ASK-CTL library loaded, the use verification is performed. To do this, it is necessary to specify the properties we want to prove. At this moment the designer is asked for the file name with the specifications. After the designer indicates this file name, the model checking is performed.
An important task in verifying systems is to identify and define properties to be proved. As the goal is to prove properties for the architecture and for the interface of the components, the framework model should be used regardless the internal component details to prove properties about the whole system. In order to do that, a technique to identify functional system scenarios is used to define interesting properties to prove. To illustrate the scenarios

Message Sequence Charts (MSC) are used. MSCs are automatically generated during the simulation of the HCPN model. The verification strategy defined in this work consists of the following steps:

1. Identification of scenarios;
2. Automatic MSC generation for each scenario based on simulation;
3. Properties identification based on the MSCs and scenarios;
4. Atomic propositions and formula specifications in temporal logic;
5. Model checking.

When performing the use verification activity of the reuse process described in Section 3, only steps 4 and 5 are done. The strategy above is a more generic approach to verify models of systems, specifically for planning and flow properties. In the rest of this section the verification is considered as the more general approach, but the reader must have in mind that the use verification activity for reused models of specific components are captured by steps 4 and 5.

Suppose that the devices send an initial signal when the system is turned on or a new device is plugged in. Also, suppose that the system perform some task when it receives this kind of message and send back to the device an acknowledgement, calibration, or even an initialization message. Such scenario is illustrated in Fig. 11.

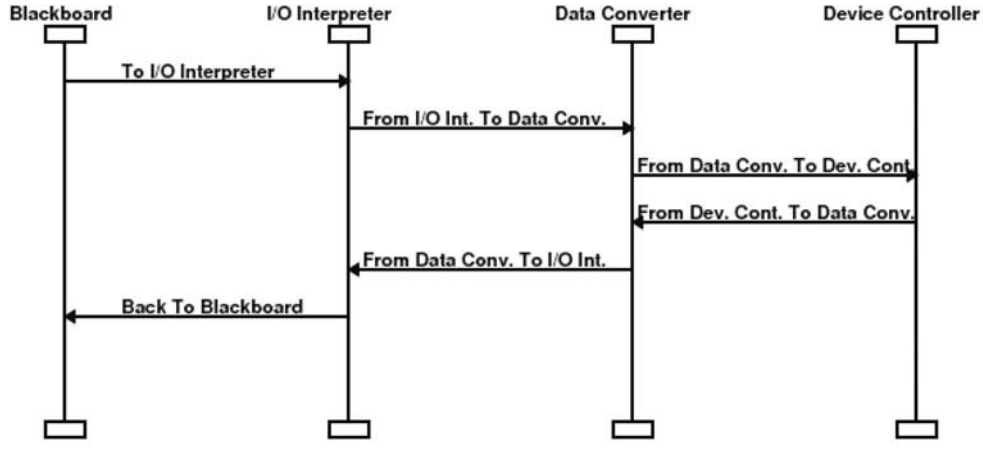


Fig. 11. Data converter flow to control signal.

When a device sends an initialization signal, the data converter sends it to the device controller to perform the associated control tasks. Since a MSC diagram is generated based on simulation it captures only a single execution sequence for the information flow in the model. It might be the case that there is some situation where this expected sequence or flow is violated. Therefore, the scenario for all possible model behavior must be verified performing model checking to guarantee that the expected flow is always respected. Considering again the scenario shown in Fig. 10, the property that when an initialization message is sent by some device the flow should be through the device controller must be provedL The following ML code fragment specifies atomic propositions and the temporal formula to prove this property.

```
fun PA=Mark.CommServer'IODConIn=(ini);
fun PB=Mark.CommServer'DConOutDevC=(ini);
val formula = AND(PA,EV(PB));
```

The proposition *PA* is true if there is a token in place lODConln of page CommServer. The proposition *PB* is to true if there is a token in place DConOutDevC of page CommServer. Therefore, the formula is true if PA is true and eventually PB is true. It means that if there is a token in the data converter input, this token is sent to device controller input, which is the component that implements control tasks such as initialization, calibration, and changing devices working parameters. The evaluation of this formula to true means that this part of the model behaves as expected for all possibilities of model execution. We can proceed with the same reasoning to prove that the flow of information back to device also behaves as expected for all possibilities.

The strategy illustrated above can be used for several different properties. It is used together with the framework, and the reuse of CPN models process to promote a formal and systematic approach for the specification and verification of systems.

8. Concluding remarks

In this chapter a component-based software development approach using Coloured Petri Nets (CPN) was presented. Components promote the software development using building blocks as in other engineering disciplines. In order to develop component-based systems it is necessary to define a process.

The implementation of a reuse-based modeling process for CPN, including all the activities in a fully automatic way was introduced. The model reuse activities considered were adaptation, integration and use verification. For different application domains only the integration step is affected because of the need to define different integration strategies and a specific framework. Due to the use of the framework concept, the designer does not need to explicitly care about the model structure.

The application of the introduced methodology for the design of dependable software is natural in a context in which adaptation means refinement. The introduced approach is very useful whenever the CPN model is used in order to provide control information to some other system. The approach introduced in this work has been applied to the embedded systems domain. The adaptability is achieved in a controlled, formal and systematic way. Therefore, the process can be used in the context of critical and dependable systems, such as embedded systems. The two major problems related to the software development for embedded systems, namely specification and verification, can be effectively tackled based on the approach introduced in this chapter.

One extension of this work is to diversify the repository. This can be done in several ways. One way is to have several models for a domain and even several domains. Examples of other possible domains are communication systems, and integrated circuits. Another way is to put in the repository other kinds of artifacts like code segments, and other modeling formalisms. Furthermore, it is also needed to be able to derive source code from the CPN models. One possible idea to do this is to associate pieces of code (in C or Java) to the transitions of the models. Also, distributed repositories can be considered.

10. References

[Cheng et al., 1997] Cheng, A., Christensen, S., and Mortensen, K. H. (1997). Model checking Coloured Petri Nets exploiting strongly connected components. Technical report, Computer Science Department, Aarhus University, Aarhus (Denmark).

[Christensen and Haagh, 1996] Christensen, S. and Haagh, T. B. (1996). Design/CPN Overview of CPN ML Syntax. University of Aarhus, 3.0 edition.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). Model Checking. The MIT Press, Cambridge, MA, USA.

[Clarke and Wing, 1996] Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. ACM Computing Surveys, 28.

[Crnkovic, 2001] Crnkovic, I. (2001). Component-based software engineering - new challenges in software development. Software Focus, 2(4):127-133.

[Crnkovic and Grunske, 2007] Crnkovic, I. and Grunske, L. (2007). Evaluating dependability attributes of component-based specifications. In ICSE Companion, pages 157-158.

[Crnkovic et al., 2002] Crnkovic, I., Hnich, B., Jonsson, T., and Kiziltan, Z. (2002). Specification, implementation, and deployment of components. Communications of the ACM, 45(10):35-40.

[Des, 2006] Design/CPN 4.0. Meta Software Corporation and Department of Computer Science, University of Aarhus, Aarhus, Denmark. On-line version: http://www.daimi.aau.dk/ designCPN/.

[Donatelli and Thiagarajan, 2006] Donatelli, S. and Thiagarajan, P. S., editors (2006). Petri Nets and Other Models of Concurrency - ICATPN 2006, 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, Turku, Finland, June 26-30, 2006, Proceedings, volume 4024 of Lecture Notes in Computer Science. Springer.

[Emerson, 1990] Emerson, E. A. (1990). Temporal and modal logic. In Handbook of Theoretical Computer Science, volume B, chapter 16, pages 995-1072. Elsevier Science Publisher B.V.

[Gorgônio and Perkusich, 2002] Gorgônio, K. C. and Perkusich, A. (2002). Adaptation of Coloured Petri Nets models of software artifacts for reuse. In Gacek, C., Software Reuse: Methods, Techniques and Tools. VII International Conference on Software Reuse, number 2319 in Lecture Notes in Computer Science, pages 240-254, Austin, Texas (USA). Springer-Verlag.

[Gorton et al., 2006] Gorton, I., Heineman, G. T., Crnkovic, I., Schmidt, H. W., Stafford, J. A., Szyperski, C. A., and Wallnau, K. C, editors (2006). Component-Based Software Engineering, 9th International Symposium, CBSE 2006, V"aster°as, Sweden, June 29 - July 1, 2006, Proceedings, volume 4063 of Lecture Notes in Computer Science. Springer.

[Jensen, 1992] Jensen, K. (1992). Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, volume 1 of EACTS - Monographs on Theoretical Computer Science. Springer-Verlag.

[Jensen, 1997] Jensen, K. (1997). Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, volume 2 of EACTS - Monographs on Theoretical Computer Science. Springer-Verlag.

[Jensen, 2005] Jensen, K., editor (2005). Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, volume PB-576, Aarhus (Denmark). DAIMI.

[Jensen, 2006] Jensen, K., editor (2006). Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, volume PB-579, Aarhus (Denmark). DAIMI.

[Kleijn and Yakovlev, 2007] Kleijn, J. and Yakovlev, A., editors (2007). Petri Nets and Other Models of Concurrency - ICATPN 2007, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings, volume 4546 of Lecture Notes in Computer Science. Springer.

[Knight, 2001] Knight, J. C. (2001). Dependability of embedded systems. In Proceedings of the 23rd International Conference on Software Engineering, page 688.4. IEEE Computer Society.

[Knight, 2002] Knight, J. C. (2002). Dependability of embedded systems. In Proceedings of the 24th International Conference on Software Engineering, pages 685-686. ACM Press.

[Knight, 2004] Knight, J. C. (2004). An introduction to computing system dependability. In Proceedings of the 26th International Conference on Software Engineering, pages 730-731. IEEE Computer Society.

[Land and Crnkovic, 2007] Land, R. and Crnkovic, I. (2007). Software systems in-house integration: Architecture, process practices, and strategy selection. Information & Software Technology, 49(5):419-444.

[Lee, 1999] Lee, E. A. (1999). Embedded software - an agenda for research. Technical Report UCB/ERL No. M99/63, University of California at Berkeley.

[Lee, 2002] Lee, E. A. (2002). Embedded software. In Zelkowitz, M., editor, Advances in Computers, volume 56. Academic Press, London (UK).

[Lemos and Perkusich, 2001] Lemos, A. J. P. and Perkusich, A. (2001). Reuse of Coloured Petri Nets software models. In Proc. of The Eighth International Conference on Software Engineering and Knowledge Engineering, SEKE'Ol, pages 145-152, Buenos Aires (Argentina).

[Long, 1993] Long, D. L. (1993). Model Checking, Abstraction, and Compositional Reasoning. PhD thesis, Carnegie Mellon University.

[McMillan, 1993] McMillan, K. L. (1993). Symbolic Model Checking. Kluwer Academic Publishers, Boston/ Dordrecht/ London.

[Murata, 1989] Murata, T. (1989). Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4):541-580.

[Nierstrasz et al., 2002] Nierstrasz, O., Arevalo, G., Ducasse, S., Wuyts, R., Black, A. P., Miiller, P. O., Zeidler, C, Genssler, T., and van den Born, R. (2002). A component model for field devices. Lectures Notes in Computer Science, 2370:200-216.

[Nierstrasz et al., 1992] Nierstrasz, O., Gibbs, S., and Tsichritzis, D. (1992). Component-oriented software development. Communications of the ACM, 35(9):160-165.

[Peled, 1994] Peled, D. (1994). Combining partial order reductions with on the fly model checking. In CAV '94: Proceedings of the 6th International Conference on Computer Aided Verification, pages 377-390, London, UK. Springer-Verlag.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77), pages 46–57, Providence, Rhode Island. IEEE, IEEE Computer Society.

[Ramadge and Wonham, 1987] Ramadge, P. J. G. and Wonham, W. M. (1987). On the supremal controllable sublanguage of a given language. SLAM Journal on Control and Optimization, 25(3):637-659.

[Ramadge and Wonham, 1989] Ramadge, P. J. G. and Wonham, W. M. (1989). The control os discrete event systems. Proceedings of the IEEE, 77(l):81-97.

[Silva and Perkusich, 2005] Silva, L. D. and Perkusich, A. (2005). A Model-Based Approach to Formal Specification and Verification for Embedded Systems Using Coloured Petri Nets. In: Colin Atkinson; Christian Bunse; Hans-Gerhard Gross; Christian Peper. (Org.). Component-Based Software Development for Embedded Systems: An Overview on Current Research Trends. Berlin: Springer-Verlag, v. 3778, p. 35-58.

[Szyperski, 1999] Szyperski, C. (1999). Component Software: Beyond Object-Oriented Programming. Addison-Wesley.

[Ullman, 1998] Ullman, J. D. (1998). Elements of ML Programming. Prentice Hall, 2 edition.

[Valmari, 1991] Valmari, A. (1991). A stubborn attack on state explosion. In CAV '90: Proceedings of the 2nd International Workshop on Computer Aided Verification, pages 156-165, London, UK. Springer-Verlag.

[van Steen et al., 1998] van Steen, M., van der Zijden, S., and Sips, H. (1998). A view on components. In Proceedings of the 9th International DEXA Workshop on Database and Expert Systems Applications, IEEE Computer Society, Los Alamitos, California.

22

Formalizing and Validating UML Architecture Description of Service-Oriented Applications

Zhijiang Dong[1], Yujian Fu[2], Xudong He[3] and Yue Fu[4]
[1]*Department of Computer Science, Middle Tennessee State University*
[2]*Department of Computer Science, Alabama A&M University*
[3]*School of Computer Science, Florida International University*
[4]*Department of Technology, Fuyuan High Technology Co.,Ltd.*
[1,2,3]*USA*
[4]*P.R. China*

1. Introduction

Service-oriented applications, especially web systems, are self-descriptive software components which can automatically be discovered and engaged, together with other web components, to complete tasks over the Internet. The importance of service-oriented application architecture descriptions has been widely recognized in recently year. One of the main perceived benefits of a service-oriented application architecture description is that such a description facilitates system property analysis and thus can detect and prevent web design errors in an earlier stage, which are critical for service-oriented applications. Software architecture description and modeling of a service-oriented application plays a key role in providing the high level perspective, triggering the right refinement to the implementation, controlling the quality of services of products and offering large and general system properties. While several established and emerging standards bodies (e.g., [5, 4, 3, 1, 2] etc.) are rapidly laying out the foundations that the industry will be built upon, there are many research challenges behind service-oriented application architecture description languages that are less well-defined and understood [33] for the large number of web service application design and development.

On the other hand, Unified Modeling Language (UML), a widely accepted object-oriented system modeling and design language, has been adapted for software architecture descriptions in recent years. Several research groups have used UML extension to describe the service-oriented application's architecture ([7, 29]). However, it is hard to detect the system problems, such as correctness, consistency [30] etc., of the integration of Web services without a formal semantics of web services architecture. Currently, although a software architecture description using UML extension contains multiple viewpoints such as those proposed in the SEI model [39], the ANSI/IEEE P1471 standard, and the Siemens [31]. The component and connector (C&C) viewpoint [42], which addresses the dynamic system behavioral aspect, is essential and necessary for system property analysis.

To bridge the gap between service-oriented application architecture research and practice, several researchers explored the ideas of integrating architecture description languages (ADLs) and UML [8, 13, 14, 35]. Most of these integration approaches attempted to describe elements of ADLs in terms of UML such that software architectures described in ADLs can be easily translated to extensions of UML. There are several problems of the above approach that hinder their adoption. First, there are multiple ways to describe ADLs in terms of UML [24], each of which has advantages and disadvantages; thus the decision on which extension of UML to use is not unique. Second, modifications on UML models are difficult to be reflected in the original ADL models since the reverse mapping is in general impossible. Finally, the software developers are required to learn and use specific ADL to model software architecture and use the specific extension of UML, which is exactly the major cause of preventing the wide use of ADLs. Currently, there is less work involved to apply these methodologies to the service-oriented applications.

In this paper, we present an approach opposite to the one mentioned above and apply our approach to the web applications, i.e. we translate a UML architecture description into a formal architecture model for formal analysis. Using this approach, we can combine the potential benefits of UML's easy comprehensibility and applicability with a formal ADL's analyzability. Moreover, this approach is used to formally analyze the integration of web services. The formal architecture model used in this research is named SO-SAM, an extended version of SAM [27], which is based on Petri nets and temporal logic; and supports the analysis of a variety of functional and non-functional properties [28]. Finally, we validate this approach by using model checking techniques. This approach presents an effective way of the Service-Oriented Architecture (SOA) in a logical format so that stake holders can better use artifacts to leverage Unified Modeling Language (UML) components in their architecture and design efforts.

The remainder of this paper is organized as follows. In section 2, we review SO-SAM with predicate transition nets and temporal logic for high-level design. After that, we presented our approach in section 3 and the validation of the approach is demonstrated in section 4. Finally, we draw conclusions and describe future work in section 6.

2. Preliminaries

2.1 Overview of SO-SAM

SO-SAM [20] is an extended version of SAM [44] with the web service oriented features. SO-SAM [20] is a general formal framework for specifying and analyzing service-oriented architecture with two formalisms – Petri Net model and temporal logic, which is inherited from SAM.

In addition, SO-SAM extended the net inscriptions with *servicesorts* and net structure with *initial* and *final* ports that carry service triggering information. Furthermore, SO-SAM restricted SAM connector without hierarchical architecture. Finally, SO-SAM component is described by WSDL or XML. Also, the message in ports is defined by XML message. For the more information, please refer to [20]. In this paper, we choose algebraic high level nets [17] and linear time first order temporal logic as the underlying complementary formalisms of SAM. Thus, next we simply introduce the algebraic high level nets used in our approach.

2.2 SAM

SAM [44] is an architectural description model based on Petri nets [37], which are well-suited for modeling distributed systems. SAM [44] has dual formalisms underlying – Petri nets and Temporal logic. Petri nets are used to describe behavioral models of components and connectors while temporal logic is used to specify system properties of components and connectors.

SAM architecture model is hierarchically defined as follows. A set of compositions $C = \{C_1, C_2, \ldots, C_k\}$ represents different design levels or subsystems. A set of component C_{mi} and connectors C_{ni} are specified within each composition C_i as well as a set of composition constraints C_{si} , e.g. $C_i = \{C_{mi}, C_{ni}, C_{si}\}$. In addition, each component or connector is composed of two elements, a behavioral model and a property specification, e.g. $C_{ij} = (S_{ij}, B_{ij})$. Each behavioral model is described by a Petri net, while a property specification by a temporal logical formula. The atomic proposition used in the first order temporal logic formula is the ports of each component or connector. Thus each behavioral model can be connected with its property specification. A component C_{mi} or a connector C_{ni} can be refined to a low level composition C_l by a mapping relation h, e.g. $h(C_{mi})$ or $h(C_{mi}) = C_l$.

Figure 1 shows a graphical view of a simple SAM architecture model. The formal analysis and design strategy of the SAM model on the software architecture is given in work [27].

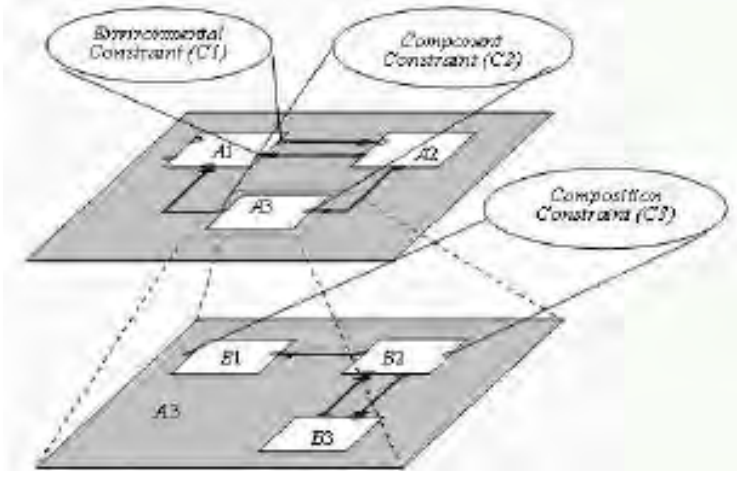


Fig. 1. A SAM Architecture Model

SAM gives the flexibility to choose any variant of Petri nets and temporal logics to specify behavior and constraints according to system characteristics. In our case, Predicate Transition (PrT) net [25] and linear temporal logic (LTL) are chosen.

In summary, although our work was strongly influenced by SAM, we have enhanced the state of the art by supporting modern software engineering philosophies equipped with component-based and object-oriented notations and applied to web services-oriented systems, as well as integrated with WSDL and XML.

2.3 Algebraic high-level nets

An algebraic high-level net [17] integrates Petri net with inscription of an algebraic specification defining the data types and operations. Instead of specifying a single system

model, an algebraic Petri net represents a class of models that often differ only in a few parameters. Such a compact parameterized description is unavoidable for modular specification and economic verification of net models in the dependable system design.

Generally speaking, an algebraic high-level (AHL) nets $N = (SPEC, A, X, P, T, W^+, W^-, cond, type)$ consists of following parts:

- An algebraic specification $S\ PEC = (S, OP, E)$, where $SIG = (S, OP)$ is a signature, and E is a set of equations over SIG;
- A is an $S\ PEC$ algebra;
- X is an family of S-sorted variables;
- P is a set of places;
- T is a set of transitionssuch that $P \cap T = \varnothing$;
- Two functions W^-, W^+ assigning to each $t \in T$ an element of the free commutative monoid[1] over the cartesian of P and terms of $SPEC$ with variables in X.
- $cond$ is a function assigning to each $t \in T$ a finite set of equations over signature SIG.
- $type$ is a function assigning to each place a sort in S.

Fig. 2 shows an algebraic high-level net of sender-receiver. Its algebraic specification is defined as following:

```
SPEC =
    sorts: nat, bool, data, queue
    opns: err: → data
          nil: → data
          inq: data × queue → queue
          deq: queue → queue
          first: queue → data
          empty: queue → bool
          length: queue → nat
    eqns: deq(nil) = nil
          deq(inq(x,nil)) = nil
          deq(inq(x,inq(y,q))) = inq(x,deq(y,q))
          first(nil) = err
          first(inq(x,nil)) = x
          first(inq(x,inq(y,q))) = first(inq(y,q))
          empty(nil) = true
          empty(inq(x,q)) = false
          length(nil) = 0
          length(inq(x,q)) = length(q) + 1
```

From the figure, we can see transition *send* is enabled if place $p1$ contains a data and the queue in place p has space. As a result of the firing of *send*, the data is added to the queue. Whenever place $p4$ has a token and the queue in p is not empty, transition *receive* is enabled. When it fires, the first data in the queue is output to place $p3$ and the first data in the queue is removed.

[1] A set M with an associative operation _ and an identity element for that operation is called a monoid. A commutative monoid is a monoid in which the operation is commutative. A commutative monoid is a free commutative monoid if every element of M can be written in one and only one way as a product (in the sense of _) of elements of subset P M.

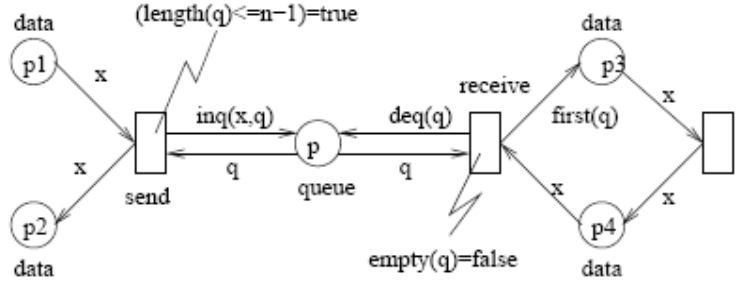


Fig. 2. Sender-Receiver Model by Algebraic High-Level Net

2.4 Linear temporal logic

Temporal formulas are built from elementary formulas using logical connectives $\neg$ and $\wedge$ (and derived logical connective $\vee$, $\Rightarrow$, and $\Leftrightarrow$, universal quantifier $\forall$ (and derived existential quantifier $\exists$), and temporal operators always $\Box$, future $\Diamond$, and until $\mathcal{U}$.

The semantics of temporal logic is defined on behaviors (infinite sequences of states). The behaviors are obtained from the execution sequences of petri nets where the last marking of a finite execution sequence is repeated infinitely many times at the end of execution sequence. For example, for an execution sequence M_0, ,, M_n, the following behavior $\sigma = << M_0, \ldots, M_n, M_n, \ldots >>$ is obtained, where M_i is a marking of the Petri net.

Let $\sigma = << M_0, M_1, \ldots >>$ be the behavior, where each state M_i provides an interpretation for the variables mentioned in predicates. The semantics of a temporal formula p in behavior σ and position j is denoted by (σ, j) j= p. We define:

- For a state formula p, $(\sigma; j) \models p \Leftrightarrow M_j \models p$;
- $(\sigma; j) \models \neg p \Leftrightarrow (\sigma, j) \not\models p$;
- $(\sigma; j) \models p \vee q \Leftrightarrow (\sigma, j) \models p$ *or* $(\sigma, j) \models q$;
- $(\sigma; j) \models \Box p$, $(\sigma, i) \models p$ for all $i \geq j$;
- $(\sigma; j) \models \Diamond p$, $(\sigma, i) \models p$ for some $i \geq j$;
- $(\sigma; j) \models p \mathcal{U} q \Leftrightarrow \exists i \geq j : (\sigma, i) \models q$, *and* $\forall j \leq k < l, (\sigma; k) \models p$.

2.5 Component and connector view

Component and connector view was one of the four views proposed in [31, 32], which is described as an extension of UML. The component and connector view describes architecture in terms of application domain elements. In this view, "the functionality of the system is mapped to architecture elements called components, with coordination and data exchange handled by elements called connectors." [31]

In the component and connector view, components, connectors, ports, roles and protocols are modelled as UML stereotyped classes. Each of them is represented by a special type of graphical symbol, as summarized in Fig. 3. A component communicates with another component of the same level only through a connector by connections, which connect relevant ports of components and roles of connectors that obey a compatible protocol. In addition to the connections between components and connectors, ports (roles, resp.) of a component (connector, resp.) can be bound to the ports (roles, resp.) of the enclosing

component (connector, resp.).

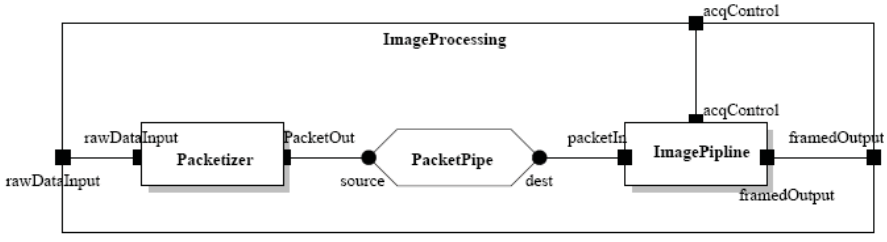
In order to present our approach we use an image processing example used in the distributed web application that was adapted from [31]. Fig. 4, 5, 6 from [31] shows a concrete and complete component and connector view, which is the running example of this paper. Fig. 4(a) is a configuration of *ImageProcessing* component. Fig. 4(b) shows another aspect of the configuration. Both of them are UML class diagrams and model different aspects of the system. From these two figures, we can see the component *ImageProcessing* contains two components: *Packetizer* and *ImagePipline*, and one connector *Packet-Pipe*. The ports of component *ImageProcessing*, *raw-DataInput*, *acqControl*, and *framedOutput* are bound to ports *rawDataInput* of component *Packetizer*, *acqControl* and *framedOutput* of component *ImagePipeline* respectively. Component *Packetizer* communicates with component *ImagePipeline* through connector *PacketPipe*. Component *Packetizer* and connector *PacketPipe* is connected by a connection between port *PacketOut* and role *source*, which obey (conjugate) protocol *DataPacket*. Component *ImagePipeline* and connector *PacketPipe* is connected by a connection between port *PacketIn* and role *dest*, which obey (conjugate) protocol *RequestDataPacket*. Being a conjugate means that the ordering of the messages is reversed so that incoming messages are now outgoing and vice versa.

Element/Relation	UML Notation	Graphical Symbol
component	Class <<*component*>>	
connector	Class <<*connector*>>	
role	Class <<*role*>>	
port	Class <<*port*>>	
protocol	Class <<*protocol*>>	
composition	Composition	
binding	Association	
connection	Association	
obeys	Association	obeys
obeys conjugate	Association	obeys conjugate

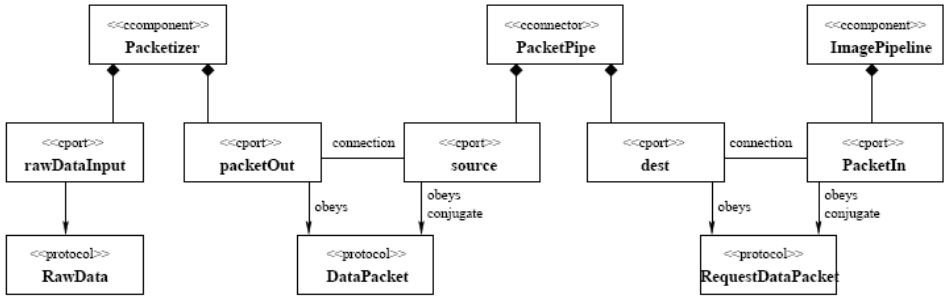
Fig. 3. UML Extension for component and connector View

Fig. 4. alone is not enough to illustrate component and connector view since only components and connectors of the system and corresponding connections among them are demonstrated. Additional diagrams are needed to define protocols and functional behavior of components and connectors. A protocol, represented by a stereotyped class, is defined as

a set of incoming message types, a set of outgoing message types and the valid message exchange sequences. The valid message exchange sequence is represented by a sequence diagram. Fig. 5 shows the definition of *RawData*, *DataPacket*, and *RequestDataPacket* protocols. From Fig. 5(c), we can see protocol *RequestDataPacket* has one incoming message: *packet(pd)*, and three outgoing messages: *subscribe(c)*, *desubscribe(c)*, and *requestPacket(c)* where *c*; *pd* are parameters of messages. In order to communicate with object *B* based on protocol *RequestDataPacket*, object *A* first sends object *B* a message *subscribe(c)* where *c* indicates the sender *A*. Then a message *requestPacket* is sent to *B* to request a packet. Later, object *A* may receive a packet *pd* from *B*. The symbol "*" in the figure indicates that the pair of message *requestPacket* and *packet(pd)* may occur many times. Finally, object *A* sends a message *desubscribe(c)* to *B* to stop requesting packet.



(a) Component ImageProcessing

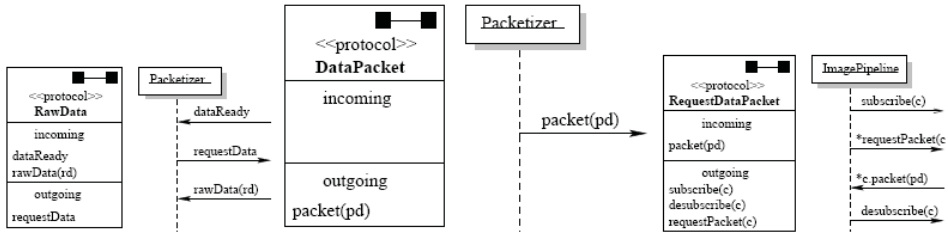


(b) Protocols for Packetizer and PacketPipe

Fig. 4. Structural Aspect of Component and Connector View

The behavior of components/connectors may be described formally by UML statechart diagrams, for example the behavior of component *Packetizer* and connector *PacketPipe* in Fig. 6. Statechart diagrams describe the dynamic behaviors of objects of individual classes through a set of conditions, called states, a set of transitions that are triggered by event instances, and a set of actions that can be performed either in the states or during the firing of transitions. From Fig. 6(b), we can see the statechart diagram of connector *PacketPipe* contains two states: *waiting* and "*assign packet to ready client*" and seven transitions. When connector *PacketPipe* receives an event *subscribe(c)*, it invokes its operation *AddClient(c)* although we do not know exactly the functionality of this operation. When the connector receives an event *packet(pd)*, it saves the packet *pd*. And the response of connector *PacketPipe* to an event *requestPacket(c)* is up to the condition: the client *c* has read current packet or not. If yes, the connector treats it as a request for next packet; otherwise it sends current packet to client *c* through an event *c.packet*(*pd*). If all clients have read current packets, the connector updates its packet queue and enter state *"assign packet to ready*

client" in which the connector sends current packet to clients that has submitted their requests. If all requests are processed, the connector returns to state *waiting*. As we can see from this figure, connectors and components mainly handle incoming messages of protocols they obey (conjugate).



(a) Protocol RawData (b) Protocol DataPacket (c) Protocol RequestDataPacket

Fig. 5. Protocols in Component ImageProcessing

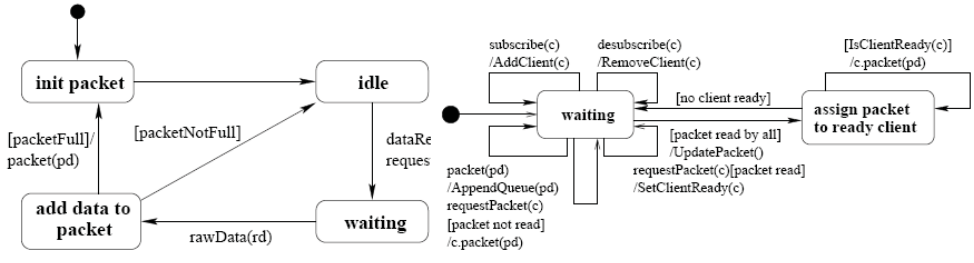
2.6 Service oriented software architecture model

Definition 1 (Web Service) *A web service is defined by a service component (composition of sub-services) as a tuple such that* $SN =< S_{ID}, f, Pt, ST >$ *where*

- S_{ID} *is the service identification. When a service component is a composition of sub-services, each sub-service has its service id and service component's id.*
- *f is SO-SAM structure mapping function.*
- *and* $Pt_{ini} \cup Pt_{f\ nl} = Pt$. *A service net does not have internal ports compared to service component, i.e.,* $Pt_{internal} = \varnothing$.
- *S T is a set of service constraints.*

The behavior of each web service S_i in SO-SAM is defined by a service net SN, which must starts when the initial ports Pt_{ini} has messages and ends when the final ports receive messages. The properties are defined using a set of temporal logic formulae ST. The relation between service net and the behavior model of a service component can be summarized as

- A service net SN is a subset of the behavior model of a service component $SNcm$, i.e., $SN \_ SNcm$. A service net can be a single activity of a service component that describes only a sub-service of that service component, or a composite service of all subservices of that service component. The relation between a service net and other service nets of a service component can be publishing, binding, discovery, integration, etc.



(a) Behavior of Component Packetizer (b) Behavior of Connector PacketPipe

Fig. 6. State Chart Diagram of Elements

- Ports of a service net is a subset of ports of its service component, i.e., $Pt \in Pt_{Cm}$.
- If a service net SN is the behavior model of a service component SN_{Cm}, then the service component does not have internal ports, i.e., $Pt_{internal} = \varnothing$.
- Set of service constraints of a service net ST is a subset of service constraints of its service component, i.e., $ST \subseteq ST_{Cm}$.

Definition 2 (SO-SAM) *Service-Oriented Software Architecture Model, SO-SAM, is an extended version on the SAM, and defined by a set of service components* C_{Sm} *, a set of connectors* C_n*, a set of service constraints* C_{Ss} *and a structure mapping function f , i.e., SO-SAM ,* $< C_{Sm}, C_n, C_{Ss} >$.

2.7 Net specification

In the SO-SAM model we define three different group of sorts for three purposes: service description and publishing (SDP), service communication and binding (SCB), and service discovery or finding (SDF). In the service description and publishing (SDP) group, we identify four sorts in the net specification of a PrT net as portSpec, msg-Parameter, connection, and operation. Message is to specify the data identification flow through a port. Operation is to describe the operation can be imposed on messages. PortSpec is used to specify the parameters and functions of the service output from a port. Connection is to describe the protocol that used for the data flow through the port (which can be described by SOAP or HTTP). This group of sorts can be mapped to WSDL specification in the Table 1.

SDP in SO-SAM	WSDL	Description
portSpec	portType	The functions performed by the web services
msgParameter	message	The messages used by the web services
connection	binding	The communication protocols used by the web services
operation	operation	The operations can be performed on the message

Table 1. Mapping Relation between SDP (in SO-SAM) and WSDL

In the service discovery or finding (SDF) group, we concern two participants – service provider (SP) and service registry and broker (SR). Service provider has to have identification, contact info, category, service description, and so on. Service registry provides access point, communication protocol, and information about the company itself, including contact information, industry categories, business identifiers, and a list of services. Moreover, the binding process can be defined on the above specification. Let us use symbol $S\,DF =< SP, SR >$ denote all possible sorts using for the SDF group. After identifying these sorts, we can map a sort to a tag in the UDDI specification. However, reverse mapping from UDDI tags to SDF sorts is impossible. Because some tags such as *bindingTemplate* need functional description instead of signatures.

The behavior model in a SO-SAM refers to a Petri net, in this paper we use PrT net. The binding can be formally specified by the constraint function R. Checking the sat isfiability of R is to checking the each data in a message, data type matching, protocol conformation, and so on between web services from requestor and provider. Since a web service may have multiple binding templates, a mutual exclusion choice occurs.

The group of service communication and binding (SCB) is more related to the communication protocol (SOAP/HTTP). SOAP is a simple XML based protocol to let applications exchange information over HTTP. The communication protocol constructs the connection between parties. The specific sorts for the protocol signature can be message

definition of header, body and fault, encoding style etc..
All the above three groups are called service sorts $S_S = \{S_{SDP}, S_{SCB}, S_{SDF}\}$. For instance, *SName*, *SDesc*, *portS pec*, *message*,*URL* are service sorts, where *S Name* is the name (identification) of the service, *S Desc* is the service description, *URL* is the Uniform Resource Locator. We use OPS_s to denote the operations on the service sorts. We call the sorts defined in SAM model data sorts S_D.
The extension on the signature of the sorts and operations is very convenient for the specification and modeling of the web services architecture, binding, substitution, and the composition of sub-services and their integration of legacy code.
Each architectural component is either statically or dynamically realized by a web services component. Architectural components are connected to each other via XML-based message passing through Simple Object Access Protocol (SOAP) [3]. The behavior of the connection is specified by SAM architectural connectors. The message passing mediates the interactions between architectural components via the rules that regulate the component interactions. In our model, connectors carry the tasks of service compositions. Thus our model supports both executable and dynamic web service compositions.
Ports in each component are either input ports or output ports. In the extension to web applications, ports are used to transfer messages among services, same as in SAM model. However, we regulate messages as a tuple with the information of service name, service description, location, URL, etc., so that the message carries service information.
A component is composed of the above ports that carry service information, behavior description and property specification. The behavior of a component is defined by a Petri net, which is called a service net. In the service net, tokens in a place has to have specific sort to be consistent with the above port and message definition. A basic component is one that does not have sub-components and non-empty connectors, otherwise, it is a composition. A composition is a composite service. The relation between a composition and its subcomponents and connectors is defined by a mapping function f . Mapping function f is a set of maplets from super component(connector)'s identities to sub-components'(connectors').
Service integration and composition can be done through connectors. Connectors have the same definition as in SAM. The Petri net for a connector is a regular Petri net that describe the integration and composition of services. A connector cannot be a composition.

2.8 Net structure

There is a relation between the architecture elements of a component/connector and the elements in its behavior model (a PrT net). For each port of a component/connector, we have a corresponding one place defined in its PrT net. The sort of the port is same as the sort of the corresponding place. The relation between port and place can be defined as a port-place mapping function ξ as follows.
Definition 3 (Port-Place Mapping Function) *The behavior mapping function _ is a mapping relation from the set of ports of a component or a connector C_i to the set of places of its behavior model, a PrT net N_i. Let Pt be the set of ports, and P be the set of places in the behavior model, we simply use Pt and P to represent the set of identifications of ports (places), we have, ξ: $Pt \rightarrow P$,*

1. $$\xi = \begin{cases} \exists p_i \in P: \ \forall pt_i \in Pt \\ \exists \phi(p): \ \forall \phi(pt_i) \in \phi(Pt) \end{cases}$$

2. If there is a message sent out from a port (pt_i), $M^{\cdot}(p_i) < M(p_i)$, where $f(pt_i) = p_i$; if there is a message received in a port (pt_i), $M^{\cdot}(p_i) > M(p_i)$, where $f(pt_i) = p_i$.

Obviously, this port-place mapping relation is also suitable for the SAM model. The port-place mapping function ξ connects a component/connector with its behavior model explicitly.

The behavior model of a component/connector of a SOSAM model, a PrT net, is called a service net. A service net is different from the behavior model of SAM in following aspects:

1. We identify the initial places and final places for a service net.
2. The set of sorts includes both service sorts and data sorts, i.e., $S = S_S, S_D$.
3. The behavior mapping function ξ defined a relation between a components/connector and its service net and is reversible. In other words, function ξ^{-1} exists.

Definition 4 (Port) *Ports in SO-SAM are communication interfaces of each service and graphically represented by semi-circles. Messages in ports are modeled by tokens. The sort of each token is defined by two parts, service sort S_S and data sort S_D. Data sort are the sorts defined in SAM [26].*

Thus we have sort S of a port is defined by $S \triangleq S_S, S_D$

A port may have a PrT net that associated to it by a function $\beta : pt_i \rightarrow N_{pti}$, where N_{pti} is a PrT net that used to describe the operations that can be performed on the sorts of the port pt_i.

Service sort S_S is a service description, query or binding requestor, which can include a service name, operation, description, URL, etc.. Service sort must be carried by all tokens in the service net. A token may be described or represented by a PrT net since it carries service information and some service information includes both messages and functions. If the port has an associated net N, the net is actually used to describe some of the tokens or some sort of the port. The net can be a service net. A service net is a PrT net that carries service characters, and is defined as follows.

Definition 5 (Service Net) *A service net is a Petri net defined by 8-tuple, $SN \triangleq < P, T, F, \phi, R, L, M_0, M_p >$, where*

- *P, T are finite set of places and transitions; F is flow relation: $P \times T \cup T \times P$.*
- *ϕ is sort assignment: $P \rightarrow \wp(S)$ ([26]), but sorts S are extended to carry service information.*
- *The final ports Pt_{fnl} of a service net communicate with a set of initial ports Pt_{ini} of another service net through a connector.*
- *R (guard function), L (label function), and M_0 (initial markings) follow the definitions in the paper [26].*
- *M_p is place mapping function, $M_p : P_t \rightarrow P$*

where each sort must contain S_S . Mapping function M_p associates each port to a place in the service net. It is a one-to-one mapping function, because in the SAM model the place and ports share the same name, and two ports between upper level and lower level share the same name. It is also an onto mapping function because all places that are identified as communicators to the other service nets will not be increased for a basic service component.

2.9 Architecture structure

We first give the definition of the architecture structure mapping function ζ as follows.

Definition 6 (Architecture Structure Mapping Function) *The architecture structure mapping function is defined as a relation from a composition to the element set of its subelements. We use C_i to denote a composition, Cm_i and Cn_i denote the component and connector, Pt_{Ci} , Pt_{Cmi} , and Pt_{Cni} denotes the set of ports in a composition, component, and connector respectively. structurally, we have*

1. $\zeta = \begin{cases} Cm_i \cup Cn_i, & \text{if } Cm_i, Cn_i \in C_i \\ Pt^{Cmi} \cup Pt^{Cni}, & \text{if } Pt^{Cmi}, Pt^{Cni} \in Cm_i \cup Cn_i \wedge Cm_i, Cn_i \in C_i \end{cases}$

2. $\phi(Pt^{C_i}) \subseteq (Pt^{Cm_i} \cup Pt^{Cn_i})$.

The architecture structure mapping function ζ defines the structure relation between composition and its subcomponents and connectors. The function has two parts, one regulates the components and connectors in the composition, and another maps the ports of the composition with those of components and connectors. The constraints mapping can be considered in the behavior mapping. Since the behavior description of SAM architecture model is available in the bottom level of the hierarchy, we inherit this character from SAM directly without any update. Some service sorts Ss can be more abstract in the higher level abstraction. The result services after discovery and matching of these service descriptions can be satisfied with the service from requestor if there is more detailed information provided and discovered.

Definition 7 (SO-SAM Structure Mapping Function) *The mapping function f defined for SO-SAM model between two levels is the composition of behavior mapping function and architecture structure mapping function, i.e.,*

$f = \xi \circ \zeta$.

Considering the fact that behavior description is only available in the bottom level (which is inherited from SAM), this structure mapping function is also suitable for the mapping relation of SAM model.

Definition 8 (Service Component) *Each component C_{mi} in SO-SAM is defined by a tuple, component name Cm_iID, mapping function f , set of ports Pt that is composed of the set of input ports Pt_I and the set of output ports Pt_O, the set of initial ports $Pt_{ini} \in Pt$, the set of finial ports $Pt_{fnl} \in Pt$, a service net SN, and a set of temporal logic formulae ST, e.g., $Cim \triangleq < Cim\ ID, f, Pt_{ini}, Pt_{fnl}, Pt_{internal}, SN, S\ T >$.*

Initial ports are represented by dash line bold half circles, and final ports are represented by solid line bold half circles. Each set of initial ports in a service component must connect to a set of final ports in another service component through a connector, and vice versa.

In the component, each service must be started from one set of initial ports, but can be ended at multiple finial ports separately. This is because a service can reach different final states but starts at the same condition.

Connector of SO-SAM model is used to but not limited to describe the following activities:

1. Service publishing. This is an advertisement process of a service provider. The descriptions of a service or update of a service description is disclosed to possible requestors. A locating of possible matching service can be done afterwards.
2. Service discovery. We consider service discovery and finding to be the process of locating candidate service providers. Service repository maintains lists of service providers categorized according to proprietary classification schemes. Service requestor located the service based on the request and service description provided. Temporal and spatial availability for all requests is demanding for a service. Request refinement does not belong to this process.
3. Service binding. Service binding is a process that based on the discovery a connection between provider and requestor is established. A protocol for the negotiation is used after discovery.
4. Service substitution. Substitution uses accurate service descriptions to allow rational

optimization of sub-services within a composition. Assume we have two services A and B, Service A may be an electronic new report and service B an electronic weather report. If we try to outsource them then difficulties arise. A may only be offered in the USA and B in Chile. Composition of them becomes useless if you live in iceland; and pretty useless too if A is available on weekdays and B only on weekends. This raises the notion of *substitutability* in the context. The subsititution one function F by another G can be done if G has weaker preconditions and stronger postconditions. Some rules can be established for service substitution by weakening preconditions.

5. Service integration. The majority of businesses today are in an extremely dis-integrated state. Years of piecemeal IT development and investment have led to a situation common in all but the smallest organizations where there are numerous non-compatible systems. Many have been developed on different platforms using different languages. Thus organizations have created numerous barriers within their own IT infrastructures. Web services define a transport method for processes defined in XML. At the core of the Web service revolution is their ability to allow code to speak to code without human intervention. In the SO-SAM model, the connector provides the formal specification that connects service components with different interfaces. The constraint function *R* of the transitions in a connector defines the required messages in the input ports and describes the messages flow to the output ports.
6. Service composition. Compositions produces tightly-coupled integration between sub-services to ensure that value is added over the sum of the individual service. The question is: if we have two trusted services *A* and *B*, after composition of *A* and *B*, we have a service *C*, is this service *C* trustable? Simply, the question is the composition of sub-services can still hold the properties of its sub-services or not. Connectors in the SO-SAM model can formally describe the composition of subservices, thus it is possible for the formal verification of composed service against service properties.

3. Transformation from component and connector view to SO-SAM

Component and connector (C&C) view [42] has been the main underlying design principle of most ADLs [36], which is also a major view type in several software architecture documentation models supporting multiple architecture views such as SEI [39] and Siemens [31]. C&C view is essential and necessary for system dependability analysis since it captures a system's dynamic behavioral aspect. SO-SAM model and component and connector view share a set of common terms such as components, connectors, and ports. Therefore it is straightforward to map them to the counterparts in SO-SAM. However, due to the meaning difference and various formal methods to describe elements' behavior, the concrete mapping procedure is not that easy. This section shows a method to construct a complete and executable SO-SAM model from a component and connector view.

A component (connector, resp.) in component and connector view is mapped to a service component (connector, resp.) in SO-SAM model. It is easy to understand from structural aspects. However, the behavior mapping is complex since different formal methods are used to model behavior. UML statechart diagrams are used to model behavior in component and connector view, contrasting with Petri net model in SO-SAM. Fortunately, our previous work [12] showed that it is possible to transform statechart diagrams to Petri net models. In UML statechart diagrams, method invocations and relationships between variables are implicit in the elements' structure. For example, in Fig. 6(a), the conditions

PacketNotFull and *PacketFull*, and relationship between variable *rd* and *pd* is not illustrated explicitly. However, such information has to be expressed explicitly in order to obtain a complete and executable Petri net. In order to bridge the gap, we utilize algebraic high level nets [17], a variant of high level Petri nets, to model behavior of elements. This method is possible because SO-SAM model does not specify a particular Petri net model as its formal foundation. We use algebraic specifications [15] to capture structures
of elements obtained from UML statechart diagrams because algebraic specifications are abstract enough that no additional information about implementation detail is assumed, and they are also powerful enough to represent implied information about components or connectors. Although the work [12] is for SAM architecture model, we can still use it and adapt it to the SO-SAM model since they share the same net structure. The main differences exist in the service sorts in the net specification, initial and final ports in the net specification and net inscription. The following rule gives us a general idea to derive components or connectors in SO-SAM.

Rule 1 (Component and Connector) *A Component (connector, resp.) in component and connector view is mapped to a service component (connector, resp.) in SO-SAM according to following steps:*

Step 1 *An algebraic specification, which specifies the abstract interface of the component (connector, resp.), is generated from a UML statechart diagram. The idea to construct algebraic specification is described later.*

Step 2 *Construct a complete and executable algebraic high level net from the UML statechart diagram according to the approach in [12] and the generated algebraic specification. There is a special place in the generated algebraic high level nets that contains element information and provides necessary information for transitions.*

Step 3 *A component (connector, resp.) with a UML statechart diagram in component and connector view is mapped to a component (connector, resp.) with an algebraic high level net in SO-SAM.*

Step 4 *A composited connector in the component and connector view is flattened and mapped to a connector in the SO-SAM model.*

While it is inherently impossible to prove the correctness of the transformation, we have carefully validated the completeness and consistency of our transformation rule. First, from structure point of view, concepts of components or connectors in component and connector view and SO-SAM are the same. Both of them support component composition, binding with enclosing element, and they can have their own behavior and communication channels–ports or roles. Therefore, the main functionalities of components or connectors in component and connector view are presented in SO-SAM counterparts. Second, algebraic specification can be used to specify modular, more specific classes [16]. Therefore, the implied information of statechart diagrams, i.e. the operations and their properties can be correct and fully specified by algebraic specifications. Since functions of algebraic specifications only define what to be done, no additional implementation information not implied in statechart diagrams is introduced. Finally, our previous work [12] and others work [41] have shown that the behavior described by statechart diagrams can be fully captured by corresponding Petri nets.

The idea to obtain algebraic specifications from UML statechart diagrams is as follows:

- For each element (component and connector), its algebraic specification defines a sort, called *element sort*, like *packetizer* for component *Packetizer* and *packetpipe* for connector *PacketPipe*. If a data type of parameters is not defined by a primitive algebraic specification, a new algebraic specification is imported. Such a imported algebraic

specification generally defines only one sort, like *Packet* for component *Packetizer*. Each action of transitions in a UML statechart diagram is considered as a function such that: *action name* : *element sort* × *parameters sort list* → *element sort*. Here *parameters sort list* includes service sorts as well. For a guard condition of a transition, a function from *element sort* (with necessary parameter sorts) to boolean is added.

- For each variable that is defined in the element (i.e. the variable is not defined in the related events), a function like *GetVariableTypeName* : *element sort* → *element sort* × *VariableType*[2] is specified. The properties of these functions can be constructed as equations if they are implied in the UML statechart diagrams, like guards *PacketNotFull* and *PacketFull* cannot hold at the same time.

From the above idea, we know that the algebraic specification for component ***Packetizer*** contains four functions, two of them correspond to guard conditions_.***PacketNotFull***(), _.***PacketFull***() : ***packetizer*** → ***bool***; and one is obtained from actions: _.***AddRawData***( ) : ***packetizer*** × ***rawdata*** → ***packetizer***, and one from undefined variable: _.***GetPacket***() : ***packetizer*** → ***packetizer***× ***packet***. In these functions, "_" is used to indicate a variable placeholder, ***bool*** is a sort defined in primitive algebraic specification Bool [15], and sorts ***rawdata*** and ***packet*** are defined in imported algebraic specifications ***Packet*** and ***RawData*** respectively, which are defined by users and normally only one sort (***rawdata***, ***packet*** resp.) is specified. Appendix A gives complete algebraic specifications of component ***Packetizer*** and connector ***Packet-Pipe*** obtained from Fig. 6.

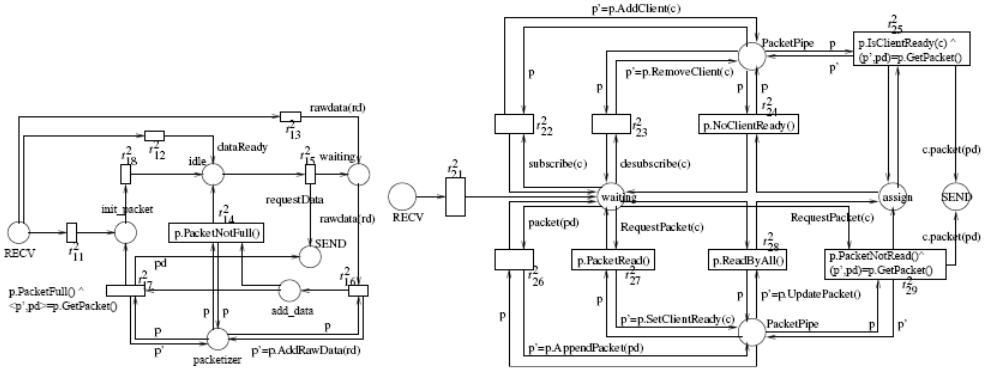
With these algebra specifications, we can generate corresponding algebraic high level nets according to Rule 1. Fig. 7 shows the generated Petri nets from UML statechart diagrams in Fig. 6. Each generated Petri net has three special places: RECV containing messages from environment, SEND temporarily storing messages generated for its environment, and the place whose name is the same as its element's name (here, ***Packetizer*** and ***PacketPipe*** resp.), holding the abstract structural information of the element. In additional to these three places, there is a corresponding place indicating current status for each state in statechart diagrams, for example, places ***idle***,***waiting***, ***init packet*** and ***add data*** for the same name states. A special token in these places indicates if the corresponding state is active. Place RECV sends events from external environment to places that are interested in the event. In Fig. 7(a) place ***idle*** and ***waiting*** are interested in event ***dataReady*** and ***rawdata(rd)*** respectively. If state ***idle*** is active and an event ***dataReady*** is available, transition t_{15} is fired. As a result, an event ***requestData*** is added to place SEND, and place ***waiting*** becomes active. State ***add data*** becomes active if state ***waiting*** is active and an event ***rawdata(rd)*** is available. At the same time, the token in place ***packetizer*** is changed to another one through operation _.***AddRawData***(_) of algebraic specification ***Packetizer***.

Components and connectors in component and connector view are connected through a connection if they are enclosed directly by the same element and the corresponding ports and roles obey (conjugate) a compatible protocol. Therefore, the mapping from ports or roles in component and connector view to ports in SO-SAM is actually the mapping from relevant protocols describing behavior of ports or roles to ports of SO-SAM components/connectors. However, ports in SO-SAM models have their own characteristics.

---

[2] This is actually the abbreviation of two functions: *GetVariable* : *element sort* → *VariableTypes* and *UpdateElement* : *element sort* → *element sort* since these two functions are invoked sequently in our example.

A port in SO-SAM model is a place that has either no incoming arcs or no outgoing arcs. In other words, the communication between ports is unidirectional. Therefore, a protocol in component and connector view, which consists of a set of incoming message types, a set of outgoing message types and the valid message exchange sequences, is mapped into a set of interface places (To avoid confusion, we use interface places to refer to ports in SOSAM model). The type of tokens in an interface place is $OID \times OID \times MESSAGE\_TYPE$, where OID is a set of unique identification number for each instance of the element, which specifies sender and receiver of a message, and $MESSAGE\_TYPE$ is the set of message types of the protocol (Here we ignore the parameters of messages for brief). Rule 2 specifies how to map a port/role in component and connector view to interface places in SO-SAM.



(a) Petri net for Packetizer (b) Petri net for PacketPipe

Fig. 7. Behavior Model of Elements

Rule 2 (Ports and Roles) *A port (role, resp.) of a component (connector, resp.) in component and connector view is mapped to a set of interface places of the corresponding component (connector, resp.) specified by Rule 1: For each protocol that the port (role, resp.) obeys (conjugate), each kind of incoming messages is mapped to an incoming (outgoing, resp.) interface place of the component (connector, resp.) with the name of the message type; and each kind of outgoing messages is mapped to an outgoing (incoming resp.) interface place of the component (connector, resp.).*

Initial and final ports can not be obtained from the UML architecture description directly. We provided two possible solutions:

- *One is extending C&C view with new UML stereotypes initialPort and finalPort. This would bring a direct transformation from C&C architecture to SOSAM. The problem is this also brings more complexity into UML architecture description.*
- *Another is manually adding the specification for these ports according to the system architecture description. For instance, we can say dataReady and RawData as initial port and frameout and final port in our case.*

A port (role, resp.) of an element is actually –roughly speaking– a "channel" that forwards messages of specified types either from element itself to environment, or from environment to element. In Rule 2, a token represents an occurrence of an message of specified type, and the direction of a message is specified by the place containing the token – incoming or outgoing. Therefore, the mapping in Rule 2 conserves the main structural features of ports/roles and related protocols, and the reverse mapping exists, which ensures the

correctness of the rule.

The behavior of a protocol, defined by UML sequence diagrams to demonstrate valid message exchange sequences, actually specifies possible sequences of relevant messages along time axle. A sequence of protocol messages illustrates their occurrence order, which can be specified by a set of temporal constraints, the basic predicates of which are the names of interface places obtained through Rule 2. For example, from Rule 2, we know port *RawData* of *Packetizer* is represented by two incoming places *dataReady* and *rawData*, and one outgoing place *requestData*. We use predicate *dataReady*(<*sid*, *rid*,$m_{dataReady}$>) to describe if place *dataReady* contains a token representing an event *dataReady* that is sent to *rid* by *sid*.

In order to construct temporal constraints, we consider two elements communicating with each other through a protocol, for example *RawData*. First we only consider a pair of adjacent events, for example *DataReady* and *requestData*. For this pair of events, it means if an event *DataReady* occurs, then an event *requestData* must occur some time later, which is described by a temporal formula:

$$\forall <sid,rid,md>, \Box(dataReady(<sid, rid,md>) \rightarrow \Diamond requestData(<sid, rid, mr>)) \quad (1)$$

However, this temporal formula cannot reflect the situation implied in the sequence diagram of the protocol: no other events of the protocol can occur between events *dataReady* and *requestData*. In order to describe this implied property, we have a reasonable assumption at architecture level that the communication media is reliable, no message is lost and no need to resend a message. Therefore, another temporal formula is introduced to address this missing situation:

$$\Diamond\forall <sid,rid,m_d>, \Box(dataReady(<sid,rid,m_d>) \rightarrow \neg((\bigcirc dataReady(<sid,rid,m_d>)) \vee$$

$$requestData(<rid,sid,m_r>) \vee rawData(<sid; rid;m_{rd}>)) \mathcal{U} requestData(<sid,rid,m_r>)) \quad (2)$$

This temporal formula means if an event of *dataReady* occurs, no other events such as *dataReady*, *requestData* and *rawData* can occur before the first event of *request-Data*. Predicate $\bigcirc$*dataReady*(<*sid*, *rid*,m_d>) is used to guarantee that the temporal formula is satisfied at the time the event *dataReady* occurs. Therefore, given a sequence diagram of a protocol with n messages, we can obtain $(n - 1) \times 2$ temporal formulas.

In addition to the consideration of one session of a protocol, we have to inspect the relationship of two adjacent sessions of the same protocol between two objects, i.e. one session can start only after the previous session ends. Such a relationship is specified by a temporal constraint:

$$\forall <sid,rid,mrd>, \Box(rawData(<sid,rid,mrd>) \rightarrow \neg(dataReady(<sid,rid,md>) \vee requestData$$

$$(<rid,sid,mr>) \vee (\bigcirc rawData(<sid,rid,mrd>))) \mathcal{U} dataReady(<sid, rid,md>)) \quad (3)$$

Although we think the above generated constraints are strong enough, there is still one more case we ignored: the first session of a protocol in a running system may starts with any messages but the first message. For example, a session of protocol *RawData* starts with message *dataReady*, and then obeys relevant part of the sequence diagram. We can see this session satisfies the above temporal formulas, but conflicts with the behavior of the protocol. Such a case can be avoided in three different ways, and the choice of them is up to users. One is to introduce a temporal predicate *basetime* that holds only at the time "zero", and a

new temporal formula:

$$\Box(basetime \rightarrow \neg(dataReady(<sid, rid, md>) \vee requestData(<rid, sid, mr>) \vee rawData(<sid, rid, mrd>)) \; U \; dataReady(<sid, rid, md>)) \quad (4)$$

The second method is to introduce a past time operator such as "eventually in the past". The final way is to prove that system structure guarantees that such case cannot happen.

Thus, from the above discussion, a sequence diagram for a protocol is mapped to a set of temporal constraints. Appendix B shows the full property constraints derived from the sequence diagrams of protocols *RawData*, *DataPacket* and *RequestDataPacket*.

The following rule is used to construct a set of constraints for components or connectors according to the above discussion.

Rule 3 (Constraint) *For each protocol that a port (role, resp.) obeys (conjugate), a set of constraints, generated from the corresponding sequence diagram according to the above discussion, is added to the property specification of corresponding components (connectors, resp.) . When a constraint is added to a component (connector, resp.), sid or rid in tokens (the choice is up to the direction of corresponding message) is substituted by the actual identification number of the component (connector, resp.) since the component (connector, resp.) can only receive messages sent to itself.*

A sequence diagram of a protocol specifies possible message communication sequences. However, it is impossible to limit the firing sequences of transitions in Petri nets to meet specified occurrence sequences of tokens in places. Although we cannot specify the firing sequences of transitions, but we can prove that if each possible firing sequence meets the behavior of a protocol. From the above discussion, we can see the generated set of temporal formulas exactly realizes the behavior of a protocol – the message sequences. By adding these temporal formulas as property specifications to components/connectors obeying the protocol, inconsistencies between behavior of elements and protocols can be easily detected. Since the behavior mapping in Rule 1 is complete and consistent, we know the detected inconsistencies also exist in the original model, i.e. Rule 3 is complete and consistent.

We may obtain a component (connector, resp.) with a behavioral model, and related ports and constraints according to Rules 1, 2 and 3 respectively. Next task is to get a complete component or connector, i.e. ports of a component or connector has to be integrated with its behavior model. Rule 4 is used to guide such a procedure, and Rule 5 establishes the connection between components and connectors.

Rule 4 (Integration) *The interface places, i.e. ports of a component (connector, resp.) in SO-SAM are integrated into its behavior model with the previous generated algebraic high level nets according to the following steps:*

Step 1 *Each incoming interface place is connected to place RECV through a transition, firing of which transmits tokens in the incoming place that are sent to the instance of component or connector to place RECV unconditionally.*

Step 2 *Each outgoing interface place is connected to place SEND through a transition, which forwards tokens of a special type in place SEND to the outgoing place.*

Rule 5 (Connection) *From Rules 2 and 4, if there is a connection between ports of a component and a role of a connector, then generated behavior models of the component and connector share a set of places that corresponds to the protocol they obey (conjugate). Therefore, to establish the connection between a component and a connector in SO-SAM, we merge these shared interface places because an incoming (outgoing, resp.) interface place in the component has an outgoing (incoming, resp.) counterpart in the connector such that they contain messages of the same type, and vice versa.*

In component and connector view, relationships between ports and behaviors are not specified explicitly. A port forwards incoming messages to the queue of the component/ connector, which provide events for its behavior – the statechart diagram. The statechart diagram sends messages to its environment through a port. In Rule 4, place SEND serves as output queue and place RECV is input queue. The forward action is represented by the firing of transitions connecting place SEND, RECV and other interface places. Therefore Rule 4 captures the communication between ports and the corresponding behaviors in component and connector view.

Due to the space limitation, we cannot specify the transformation of binding and multiplicity. However, such transformations are similar and straightforward. Fig. 8 shows the final result of generated SO-SAM model from the running example. In order to give a concise description, algebraic specifications and internal parts of behavioral models are omitted.

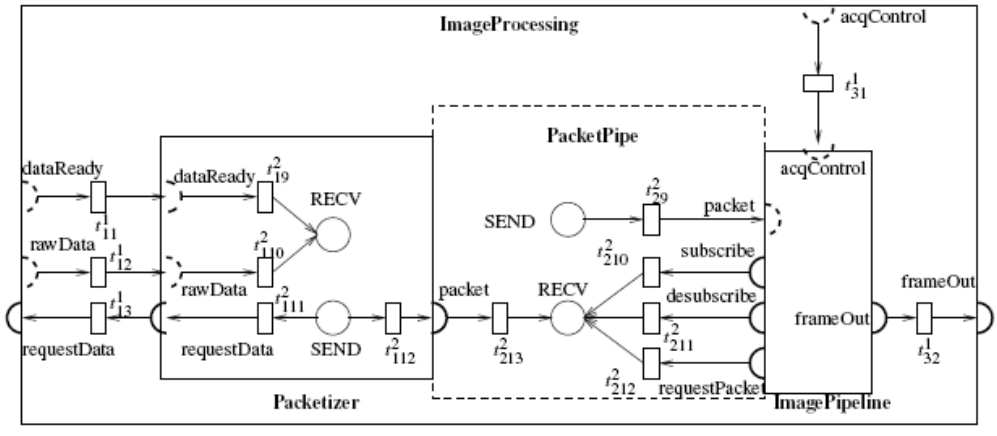


Fig. 8. ImageProcessing in SO-SAM

In Fig. 8, components, for example component *Packetizer* and component *ImageProcessing* are represented by solid rectangles, while connectors such as *PacketPipe* are represented by dashed rectangles. The Petri nets enclosed by rectangles are the behavior models of components or connectors. Semicircles on the edge of rectangles are places that represent ports derived from relevant protocols. An inside semicircle indicates an incoming place that only receives tokens from environment, while an outside semicircle indicates an outgoing place that only sends tokens to environment. For example, component *Packetizer* has two incoming places *dataReady, rawData,* and one outgoing place *requestData*. These three places are derived from protocol *RawData* according to Rule 2. Component *Packetizer* and connector *PacketPipe* is connected through Rule 5. The binding between components and its enclosing component is implemented as a transition between corresponding places, which only forwards tokens from one place to another according to types of places (i.e. incoming or outgoing). For example, transition t_{11} forwards tokens in place *dataReady* of *ImageProcessing* to place *dataReady* of *Packetizer,* while transition t_{13} forwards tokens in place *requestData* of *Packetizer* to place *requestData* of *ImageProcessing*.

Finally, we give an execution path of component *ImageProcessing*. Let component *Packetizer* be in state *idle,* connector *PacketPipe* in state *waiting,* and place *Image-Processing.dataReady* contains a token representing message *dataReady*. This initial

condition can be represented by the initial marking (ImageProcessing.dataReady, Packetizer.idle, PacketPipe.waiting). Here we only list related places (not including places such as *packetizer* and *PacketPipe*) that contain tokens, and ignore concrete token values that can be derived from context. We also assume that a packet consists of only one raw data, i.e. operation *PacketFull*() will be true if *AddRawData*() is invoked once. Table 2 shows the execution of communication based on protocols *RawData* and *DataPacket*. This example demonstrates the application of our method.

4. Validation of the approach

The SO-SAM model allows formal validation of a service net against system constraints and property specified on its abstraction represented by a component or connector. Here, validation means that the developer can animate the specification by providing initial markings and checking if the responses meet the expected results. Validation of SO-SAM is based on the precise syntax and semantics of Petri net formal language and temporal logic. The validation will cover the topology and dynamic behavior of the Petri net as well as temporal logic formulae. Here we simply introduce how to translate SO-SAM model to the Maude [9] language. For the details, please refer to the work [22].

Step	Marking of Component *ImageProcessing*	Fired Transition
1	idle, ImageProcessing.dataReady, PacketPipe.waiting	t^1_{11}
2	idle, Packetizer.dataReady, PacketPipe.waiting	t^2_{19}
3	idle, Packetizer.RECV, acketPipe.waiting	t^2_{12}
4	idle, PacketPipe.waiting	t^2_{15}
5	Packetizer.waiting, Packetize.SEND, PacketPipe.waiting	t^2_{111}
6	Packetizer.waiting, Packetizer.requestData, PacketPipe.waiting	t^1_{13}
7	Packetizer. waiting, Image Processing. request Data, PacketPipe. waiting	unspecified transition
8	Packetizer.waiting, PacketPipe.waiting	unspecified transition
9	Packetizer.waiting, ImageProcessing.rawData, PacketPipe.waiting	t^1_{12}
10	Packetizer.waiting, Packetizer.rawData, PacketPipe.waiting	t^2_{110}
11	Packetizer.waiting, Packetizer.RECV, PacketPipe.waiting	t^2_{13}
12	Packetizer.waiting, PacketPipe.waiting	t^2_{16}
13	Packetizer.add data, PacketPipe.waiting	t^2_{17}
14	Packetizer.initial packet, Packetizer.SEND, PacketPipe.waiting,	t^2_{18}
15	Packetizer.idle, Packetizer.SEND, Packetpipe.Waiting,	t^2_{112}
16	Packetizer.idle, Packetizer.packet, PacketPipe.waiting	t^2_{213}
17	Packetizer.idle, PackePipe.RECV, PacketPipe.waiting	t^2_{21}
18	Packetizer.idle, PacketPipe.waiting	t^2_{26}
	Packetizer.idle, PacketPipe.waiting	

Table 2. A Path of Executing Protocols RawData and DataPacket

4.1 Translation from SO-SAM to Maude

First, we presented a stepwised translation algorithm from SO-SAM model to Maude programming language. After that, the experimental results are illustrated.

Step 1. Translation to the functional module: generate the sorts operators used in the functional modules for the model signatures. This step translates each place, sorts, markings in a Petri net into the corresponding part in Maude's functional module.

Step 2. Translation to the system modules: there are three types of system modules, one is for the model signature that corresponds to the architecture structure and dynamic behavior of the model, one is for the mapping to the predicates, and one is for the model checking, which includes the property specification.

1. Each basic component and connector are defined as a system module (SysID) with the declaration of variables and necessary rules and operators. Each composition is specified as a system module that including its sub-components and connector that are predefined as a module. All guard conditions in a transition are a (un)conditional rule.
2. Each place is mapped to an operator in the predicate system module (SysID-PREDS). The connection between operators and predicate is established by an equation.
3. Model checking module (SysID-CHECK) is mainly for the initial marking and property specification.

In our translation, system signature such as sorts and operators are declared in the functional module. This translates the places/ports, sorts into algebra in Maude that will be used in the system modules. The dynamic semantics of Petri net can be mapped to the rewriting rules used in Maude. Computationally, the meaning of rewriting rules is to specify local concurrent transitions that can take place in a system if the pattern in the rule's lefthand side matches a fragment of the system state and the rule's condition is satisfied. In that case, the transition specified by the rule can take place, and the matched fragment of the state is transformed into the corresponding instance of the righthand side. Thus we can see an amazing match between semantics of Petri net and rewriting logic. These are theoretic aspect of the above translation algorithm.

4.2 Results

The basic requirements for the image processing in the distributed web applications are correctness, robustness and reliability. We use model checker of Maude [9] to validate the SO-SAM model obtained from UML architecture description against system properties. After studying models and the errors discovered during the model validation, two main property categories have been selected:

1. Structural properties: this kind of properties is closely related to the topology of the model. These properties can be directly verified on the SO-SAM model without animating the transactions. These properties are necessary conditions that ensure the feasibility of the state transitions. If one of them is not fulfilled, we can assert firmly that the communication between ports in UML description cannot happen.
2. Behavioral properties: the dynamic feature of these properties means that they are related to state changing of the system. The evaluation of the dynamic properties are based on the behavior description – Petri nets. Its verification is achieved on a set of places describing a possible evolution of the system. All four properties in section 3 fall in this group. The results output from Maude are true for all the above formulae. Most the above formulae are safety properties.

The results can be obtained within 10ms. It is worth to notice that the model checking technique used for the verification of system properties are only available for propositional formula. For the first order formula, it is still a challenge research topic in this area.

5. Related work

We can identify in the literature two categories of works that are mostly related to our research. The first one concerns works that modeling service oriented architecture descriptions using UML. The second one is composed of the works of formalizing the semantics of SOA in different aspects.

5.1 UML description of SOA

In the first category most use UML profiles to describe the service oriented architecture. [11] proposed UML profiles to specify functional aspects in SOA, which are defined based on the XML schema of Web Service Description Language (WSDL) [5]. The profile provides a set of stereotypes and tagged values that correspond to elements in WSDL, such as *Service*, *Port*, *Messages* and *Binding*. There is no consideration of nonfunctional aspects of web services. In work [34] a case study is presented on the investigation of the UML profile specification of SOA.

Compared to work [34] and [11], [6] proposes a UML profile to describe both functional and non-functional aspects in SOA. This work provides generic stereotypes to specify a wide range of applications. However, the semantics of this profile tend to be ambiguous. For example, several stereotypes for nonfunctional aspects (<<policy>>, <<permission>> and <<obligation>>) are intended to specify the responsibility of a service. There is no precise definition of how developers specify web applications with these stereotypes.

[29] proposes a UML profile to facilitate dynamic service discovery in SOA. This profile provides a set of stereotypes (e.g., <<uses>>, <<requires>> and <<satisfies>>) to specify relationships among service implementations, service interfaces and functional requirements. For examples, users can specify relationships in which a service uses other services, and a service requires other services that satisfy certain functional requirements. These relationship specifications are intended to effectively aid dynamic discovery of services.

[23] and [10] define UML profiles to specify service orchestration in UML and map it to Business Process Execution Language (BPEL) [1]. These profiles provide a limited support of non-functional aspects in message transmission, such as messaging synchrony. The proposed profile does not focus on service orchestration, but a comprehensive support of non-functional aspects in message transmission, message processing and service deployment.

[43] describes a UML profile for data integration in SOA. It provides data structures to specify messages so that users can build data dictionaries that maintain message data used in existing systems and new applications. The non-functional aspect of data integration is separated from functional one in this profile. Data integration can be enabled in an implementation independent manner.

There is less work on the service architecture description using UML architecture model. [40] specifies a series of service architecture patterns using UML service component. For instance, interaction service pattern describes capabilities and functions to deliver content

and data using a portal, or other related Web technologies, to consumers or users. This work infuses the component design with service building block to facilitate large scale system design. However, there is no formal reasoning of these patterns and how develop workers to use these patterns.

5.2 Formalizing SOA

In this paper we have briefly shown how UML architecture description model can be formalized using a biformalism SAM extension – SO-SAM, and the benefits that can be obtained from such formalization, namely the definition of integration and composition verifications between services, and the architecture reasoning that can bridge the differences between a priori incompatible Web services. Thus we have shown how existing formal methods can be successfully applied in the context of Web services, providing useful and practical advantages.

In addition, the formal specification of service properties using temporal logic provides us with a tool for expressing other complicated safety and liveness properties (apart from those already mentioned). In fact, any property expressed as a temporal logic formula can be considered as a sub-system specification, and therefore, checking that property on a certain web service component, would consist in reasoning the service-oriented architecture. On the other hand, having a simple formal description to describe web service architecture and integrations will allow us the application of model-checking techniques to construct (or extend) existing validation tools, as made in [19] with Promela.

Two major approaches for describing web service applications can be categorized: (a) the application oriented view of the service oriented applications or web systems (built only on the individual WSDL descriptions of the constituent web services); (b) the platform independent, architecture oriented view of service-oriented applications, which consists of different (simple) "global model" that describes how such independently defined service integration and compostion in high level abstraction.

BPEL4WS, WSFL and WSCDL are notations that use the application oriented view approach, whilst UML profile, service components, and web component are examples of the architecture oriented view approach. Application oriented view notations are in general more adaptable to each particular situation and system, but are not as amenable to web service reuse as architecture view descriptions are. Although the web service community is currently divided trying to decide which is the best approach, we argue that they can be considered as complementary tactics, rather than rivals.

The way to marry both approaches can be achieved by integrating and infusing the results from different categories, similarly like what we have discussed in this paper, mapping the UML architecture description to SO-SAM model and simply checking that the system properties defined over its constituent web services that can be replaced (in our sense), integrated or composed by their individual constituents (can be defined using an application oriented view approach). In this way, both approaches could easily co-exist.

Apart from the previous work of the authors [20, 22], there is a large amount of proposals in the literature dealing with composition, interoperation and adaptation issues in the field of Component-Based Software Engineering (CBSE), and in protocol verification in general [19]. Some of these works have been also applied to web service architecture reasoning. In cite [18], building on previous work in the field of Software Architecture by the same authors, a model-based approach is proposed for verifying Web service composition, using Message

Sequence Charts (MSCs) and BPEL4WS. In [38], and from a semantic Web point of view, a first-order logical language and Petri Nets are proposed for checking the composition of Web services. In [19], model-checking using Promela and SPIN is proposed for analysing the composability of choreographies written in WSFL. All these works deal with the (either manual or automated) simulation and analysis of Web service composites, been able to detect mismatch between their choreographies.

6. Conclusion

In this paper, we proposed a method to use SO-SAM to formally specify service-oriented application architectures modeled by an extension of UML – component and connector view. By doing so, we combine the benefit of UML – easy to comprehend and extensive tools support, and the analyzability of SO-SAM.

The cost of our methods mainly comes from three parts: the construction of algebraic specifications, the generation of algebraic high-level nets from statechart diagrams, and the creation of temporal formulas from sequence diagrams. Since an algebraic specification is used to model the implied information of statechart diagrams, generally speaking we can generate operation and sort definitions of an algebraic specification automatically, but not for the relationships among these operations. The size of a generated algebraic specification is "linear" to the size of implied information. From our previous work [12], we know the generation of Petri nets from a statechart diagram can be fulfilled automatically for most cases, and a Petri net and the corresponding statechart diagram are at the same size. The generation of temporal logic formulas from sequence diagrams can be largely automated since the generation is very simple and straightforward.

There are at least three immediate extensions to the work we have presented here. First, we intend to integrate the translation from UML architecture to SO-SAM with the mapping from SO-SAM to Maude so that some existing tool we have developed can be used for the model checking of system properties. And second, we intend to make effective use of the tools currently available for SAM model [21] to reason about the web specifications during the runtime. Finally, the translation into SO-SAM presented here must be extended in order to consider full application oriented view approach such as WSCI [4]; in particular, dealing with constructs such as correlations, transactions, properties and others, that have been omitted in this work. This extension would allow the analyzing on the more application oriented view approach using UML architecture descriptions.

7. Acknowledgments

This work is supported in part by Alabam A&M University.

8. References

[1] Business Process Execution Language for Web Services (BPEL4WS). Available from http://www.ibm.com/developerworks/library/wsbpel.

[2] DAML-S and OWL-S. Available from http://www.daml.org/services/owl-s/.

[3] Simple Object Access Protocol (SOAP), W3C Note 08. Available from http://www.w3.org/TR/SOAP/.

[4] Web Service Choreography Interface (WSCI) 1.0. Available from

http://www.w3.org/TR/2002/NOTEwsci-20020808/.

[5] Web Services Description Language (WSDL) 1.1. Available from http://www.w3.org/TR/wsdl.

[6] R. Amir and A. Zeid. A UML profile for service oriented architectures. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 192–193, New York, NY, USA, 2004. ACM Press.

[7] H. S. Bhatt, V. H. Patel, and A. K. Aggarwal. Web enabled client-server model for development environment of distributed image processing. In *Proceedings of the First IEEE/ACM InternationalWorkshop on Grid Computing (GRID'00)*, pages 135–145, London, UK, 2000. Springer-Verlag.

[8] S.-W. Cheng and D. Garlan. Mapping Architectural Concepts to UML-RT. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, June 2001.

[9] M. Clavel, F. Dur´an, S. Eker, P. Lincoln, N. Mart´ı-Oliet, J. Meseguer, and J. F. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.

[10] DeveloperWorks. UML 1.4 Profile for Software Services with a Mapping to BPEL 1.0, July 2004.

[11] DeveloperWorks. UML 2.0 Profile for Software Services, April 2005.

[12] Z. Dong, Y. Fu, and X. He. Deriving Hierarchical Predicate/Transition Nets from Statechart Diagrams. In *Proceedings of The 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2005)*, 2003.

[13] A. Egyed. Automating Architectural View Integration in UML. Technical Report USCCSE-99511, Center for Software Engineering, University of Southern California,Los Angeles, CA, 1999.

[14] A. Egyed and N. Medvidovic. Extending Architectural Representation in UML with View Integration. In *Proceedings of the 2nd International Conference on the Unified Modeling Language*, pages 2–16, October 1999.

[15] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.

[16] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Springer-Verlag, 1990.

[17] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic High-Level Nets: Petri Nets Revisited. In *Proceedings of Recent Trends in Data Type Specification, 9th Workshop on Specification of Abstract Data Types Joint with the 4th COMPASS Workshop*, volume 785 of *Lecture Notes in Computer Science*, pages 188–206. Springer, 1994.

[18] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Compositions. In *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, volume 00, page 152, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[19] X. Fu, T. Bultan, and J. Su. Formal verification of e-services and workflows. In *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web (CAiSE'02/WES'02)*, pages 188–202, London, UK, 2002. Springer-Verlag.

[20] Y. Fu, Z. Dong, and X. He. An Approach to Web Services Oriented Modeling and Validation. In *Proceedings of the 28th ICSE workshop on Service Oriented*

Software Engineering (SOSE2006), 2006.

[21] Y. Fu, Z. Dong, and X. He. A method for realizing software architecture design. In *Proceedings of the Sixth International Conference on Quality Software(QSIC'06)*, pages 57–64, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Y. Fu, Z. Dong, and X. He. Modeling, Validating and Automating Composition of Web Services. In *Proceedings of The Sixth International Conference on Web Engineering (ICWE 2006)*, 2006.

[23] T. Gardner. UML Modeling of Automated Business Processes with a Mapping to BPEL4WS. In *ECOOP Workshop on OO and Web Services*, July 2003.

[24] D. Garlan, S.-W. Cheng, and A. J. Kompanek. Reconciling the Needs of Architectural Description with Object-Modeling Notations. *Science of Computer Programming*, 44(1):23–49, July 2002.

[25] H. J. Genrich. Predicate/Transition Nets. *Lecture Notes in Computer Science*, 254, 1987.

[26] X. He. A formal definition of hierarchical predicate transition nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 212–229, London, UK, 1996. Springer-Verlag.

[27] X. He and Y. Deng. A Framework for Specifying and Verifying Software Architecture Specifications in SAM. volume 45 of *The Computer Journal*, pages 111–128, 2002.

[28] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng. Formally analyzing software architectural specifications using sam. *Journal of Systems and Software*, 71(1-2):11–29, 2004.

[29] R. Heckel, M. Lohmann, and S. Th¨one. Towards a UML Profile for Service-Oriented Architectures. In *Workshop on Model Driven Architecture: Foundations and Applications*, 2003.

[30] R. Heckel, H. Voigt, J. K¨uster, and S. Th¨one. Towards Consistency of Web Service Architectures. Available from http://www.upb.de/cs/agengels/Papers/2003/HeckelVoigtKuesterThoene-CI03.pdf.

[31] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison Wesley, 2000.

[32] C. Hofmeister, R. L. Nord, and D. Soni. Describing Software Architecture with UML. In *Proceedings of the TC2 1st Working IFIP Conference on Software Architecture (WICSA1)*, pages 145 – 160, 1999.

[33] R. Hull, M. Benedikt, V. Christophides, and J. Su. Eservices: A look behind the curtain. In *Proceedings of the International Symposium on Principles of Database Systems (PODS)*. ACM Press, June 2003.

[34] E. Marcos, V. de Castro, and B. Vela. Representing web services with UML: A case study. *International Conference on Service Oriented Computing*, 2003.

[35] N. Medvidovic, A. Egyed, and D. S. Rosenblum. Round-Trip Software Engineering Using UML:From Architecture to Design and Back. In *Proceedings of the 2nd Workshop on Object-Oriented Reengineering*, pages 1–8, September 1999.

[36] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering*, 26(1):70–93, 2000.

[37] T. Murata. Petri Nets: Properties, Analysis and Applications. *In Proceedings of the IEEE*, 77(4):541–580, 1989.

[38] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition

of web services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM Press.

[39] J. S. Paul Clements, Len Bass. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, January 2003.

[40] I. Prithvi Rao, Certified IT Architect. Using uml service components to represent the SOA architecture pattern. Available from http://www.ibm.com/developerworks/architecture/library/arlogsoa/.

[41] J. Saldhana, S. M. Shatz, and Z. Hu. Formalization of Object Behavior and Interactions From UML Models. *International Journal of Software Engineering and Knowledge Engineering*, pages 643–673, 2001.

[42] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[43] M. Vok´ac and J. M. Glattetre. Using a domainspecific language and custom tools to model a multitier service-oriented application—experiences and challenges. In C. W. Lionel Briand, editor, *Models 2005*, *Montego Bay*, *Jamaica October 2-7*, LNCS 3713, pages 492–506, Heidelberg, 2005. Springer-Verlag GmbH.

[44] J. Wang, X. He, and Y. Deng. Introducing Software Architecture Specification and Analysis in SAM through an Example. *Information and Software Technology*, 41(7):451–467, 1999. Appendix

A Algebraic specifications for *Packetizer* and *PacketPipe*

Packetizer(Packet, RawData) = Bool + Packet + RawData

sorts: packetizer

opns: _.PacketNotFull(): packetizer → bool
_.PacketFull(): packetizer → bool
_.AddRawData( ): packetizer × rawdata → packetizer
_.GetPacket(): packetizer → packetizer × packet

eqns: c ∈ packetizer
c.PacketNotFull() = ¬ c.PacketFull()

PacketPipe(Client) := Bool + Client

sorts: packetpipe

opns: ReadByAll(): packetpipe → bool
_.PacketRead(): packetpipe → bool
_.PacketNotRead(): packetpipe → bool
_.IsClientReady( ): packetpipe × client → bool
_.NoClientReady(): packetpipe → bool
_.AddClient( ): packetpipe × client → packetpipe
_.RemoveClient( ): packetpipe × client → packetpipe
_.AppendPacket( ): packetpipe × packet → packetpipe
_.SetClientReady( ):packetpipe × client → packetpipe
_.GetPacket(): packetpipe → packetpipe × packet
_.UpdatePacket(): packetpipe → packetpipe

eqns: pp ∈ packetpipe, c ∈ client, p ∈ packet
pp.IsClientReady(c) = true ⇒ pp.NoClientReady()=false

(pp.SetClientReady(c)).IsClientReady(c) = true
(pp.AddClient(c)).IsClientReady(c) = false

B Temporal constraints obtained from UML sequence diagrams

Temporal formulas for protocol *RawData*:
$\forall sid, rid,$
$\square$ (*dataReady*(<*sid*, *rid*, *md*>) $\Rightarrow \Diamond$*requestData*(<*sid*, *rid*, *mr*>))
$\square$(*dataReady*(< *sid*, *rid*, *md* >) $\Rightarrow \neg((\bigcirc$*dataReady*(< *sid*, *rid*,*md* >)) $\vee$ *requestData*(< *rid*, *sid*, *mr* >) $\vee$ *rawData*(< *sid*, *rid*, *mrd* >))U*requestData*(<*sid*, *rid*, *mr*>))
$\square$ (*requestData*(<*sid*, *rid*, *mr*>) $\Rightarrow \Diamond$*rawData*(<*sid*, *rid*,*mrd*>))
$\square$ (*requestData*(<*sid*, *rid*, *mr* >) $\Rightarrow \Diamond$(*dataReady*(< *rid*, *sid*, *md* >) $\vee$ ($\bigcirc$*requestData*(<*sid*, *rid*, *mr* >)) $\vee$
rawData(< *rid*, *sid*, *mrd* >))U*rawData*(<*rid*, *sid*, *mrd*>))
$\square$ (*rawData*(< *sid*, *rid*, *mrd* >) $\Rightarrow \Diamond$(*dataReady*(< *sid*, *rid*, *md* >) $\vee$ *requestData*(< *rid*, *sid*, *mr* >) $\vee$ ($\bigcirc$*rawData*(< *sid*, *rid*, *mrd* >)))U*dataReady*(<*sid*, *rid*, *md*>))

Temporal formulas for protocol *DataPacket*:

$\forall$(<*sid*, *rid*, *mp*>), $\square$ (*packet*(<*sid*, *rid*, *mp*>) $\Rightarrow$ *true*)

Temporal formulas for protocol *RequestDataPacket*:

$\forall sid, rid,$
$\square$ (*subscribe*(<*sid*, *rid*, *ms*>) $\Rightarrow \Diamond$*requestPacket*(<*sid*, *rid*, *mr*>))
$\square$ (*subscribe*(<*sid*, *rid*, *ms* >) $\Rightarrow \neg((\bigcirc$*subscribe*(<*sid*, *rid*, *ms* >)) $\times$ *requestPacket*(<*sid*, *rid*, *mr*>) $\times$ *packet*(<*rid*, *sid*, *mp*>) $\times$
desubscribe(<*sid*, *rid*, *md*>))U*requestPacket*(<*sid*, *rid*, *mr*>))
$\square$ (*requestPacket*(<*sid*, *rid*, *mr*>) $\Rightarrow \Diamond$*packet*(<*rid*, *sid*, *mp*>))
$\square$ (*requestPacket*(< *sid*, *rid*, *mr* >) $\Rightarrow \neg$(*subscribe*(<*sid*, *rid*, *ms* >) $\times$ ($\bigcirc$*requestPacket*(<*sid*, *rid*, *mr* >)) $\times$ *packet*(<*rid*, *sid*, *mp* >) $\times$
desubscribe(<*sid*, *rid*, *md*>))U*packet*(<*rid*, *sid*, *mp*>))
$\square$ (*packet*(< *sid*, *rid*, *mp* >) $\Rightarrow \Diamond$(*desubscribe*(< *rid*, *sid*,*md* >) $\times$ *requestPacket*(<*rid*, *sid*, *mr*>)))
$\square$ (*packet*(<*sid*, *rid*, *mp* >) $\Rightarrow \neg$(*subscribe*(<*rid*, *sid*, *ms* >) $\times$ *requestPacket*(<*rid*, *sid*, *mr*>) $\times$ ($\bigcirc$*packet*(<*sid*, *rid*, *mp*>)) $\times$
desubscribe(< *rid*, *sid*, *md* >))U (*desubscribe*(<*rid*, *sid*, *md* >) $\times$ *requestPacket*(<*rid*, *sid*, *mr*>)))
$\square$ (*desubscribe*(<*sid*, *rid*, *mp* >) $\Rightarrow \neg$(*subscribe*(<*sid*, *rid*, *ms*>) $\times$ *requestPacket*(<*sid*, *rid*, *mr* >) $\times$ *packet*(<*rid*, *sid*, *mp* >) $\times$
($\bigcirc$*desubscribe*(<*sid*, *rid*, *md*>)))U*subscribe*(<*sid*, *rid*, *md*>))

23

Music Description and Processing: An Approach Based on Petri Nets and XML

Adriano Baratè
Laboratorio di Informatica Musicale (LIM),
Dipartimento di Informatica e Comunicazione (DiCO)
Università degli Studi di Milano
Italy

1. Introduction

Music description and processing require formal tools which are suitable for the representation of iteration, concurrency, ordering, hierarchy, causality, timing, synchrony, non-determinism. Petri Nets are a tool which allows to describe and process musical objects within both analysis/composition and performing environments. To accomplish this objective, a specific extension known as Music Petri Nets was developed.

2. Music Petri nets

This Petri Net formalism can be applied to music field by associating music objects to places and music operators to transitions.
According to the definition in (Haus & Rodriguez, 1993), a *music object* may be anything that could have a music meaning and that could be thought as an entity, either simple or complex, either abstract or detailed. Such entity will present some relationship with other music objects. In a Music Petri Net, when a place containing an object receives a token, the music object is executed, i.e. played. To understand the following examples about transitions' behaviour, two simple music objects are shown in Figure 1 as notated fragments.

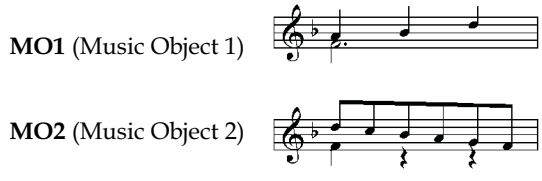


Fig. 1. Two simple music objects: MO1 and MO2

In Music Petri Nets the role played by transitions is very important: they determine – together with tokens – the evolution of the net. In our extension of Petri Nets transitions can have associated *music operators*. A transition without an algorithmic behaviour is considered having a null operator associated. When no music operators are associated, transitions are

only devoted to net evolution. Their role is dropping tokens from input places and adding tokens to output places, such as in common Petri Nets.
As stated before, when a token arrives at place with an associated music object, this object is played. In Music Petri Nets, the temporization of the execution is achieved considering the durations of the music objects (eventually) associated to the places. When a place receives one or more tokens from incoming transitions, the (eventually) associated music fragment is executed, and the tokens are blocked in the place (i.e. they cannot be transferred to outgoing transitions) until such execution is completed.
An example is provided in Figure 2, where the music object MO1 is associated to the left place with the same name, and MO2 is associated to the right one. The first measure is played when a token arrives in MO1, causing its execution. Then, only when the entire music object is played, the token is free to leave the place, and in fact it is moved to the subsequent one, originating the juxtaposed execution of the second measure. The overall result is noted in the score.

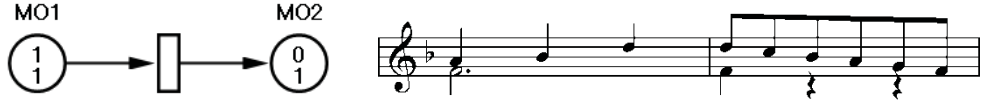
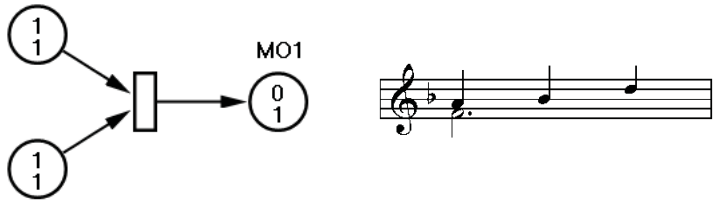
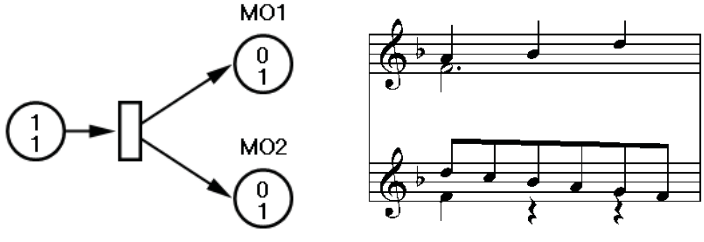


Fig. 2. The *sequence* structure

Various music structures can be created even by using transitions *without* music operators. In Figure 3 five simple nets illustrate respectively a fusion (from two objects to one object), a split (from an object to two objects), an alternative (a non-deterministic choice between two objects), and a joint structure (a logical connection between two objects).



Fusion



Split

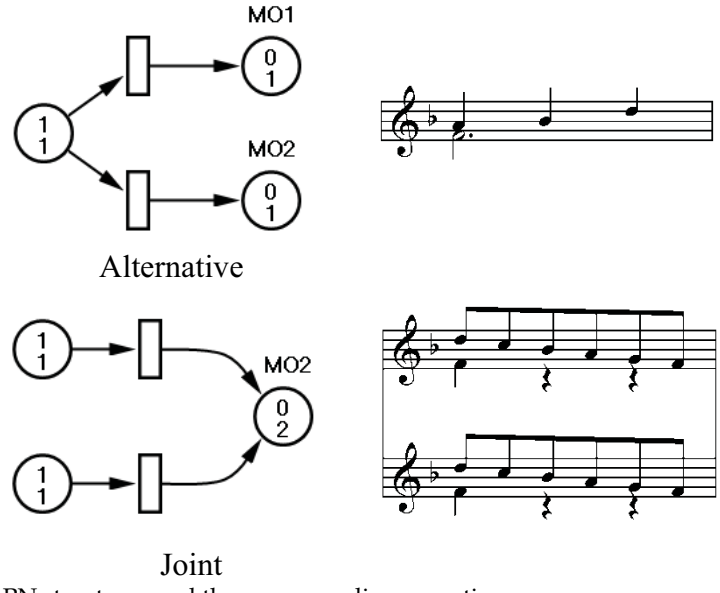


Fig. 3. Some PN structures and the corresponding executions

We have said that transitions might have associated music operators. When a music operator is specified, its purpose is applying an algorithm to change input objects (i.e. objects coming from input places), and then passing the transformed objects to output places. Typical operators associated to transitions reflect common music operators, such as inversion, retrogradation, and transposition. For example, Figure 4 shows the application of the last mentioned operator. In this net a music object is associated to the place MO3, while the right place has no associated objects. When the transition fires, it receives MO3 in input, a transposition is performed and the modified music fragment is passed to the outgoing place, that executes the new object.

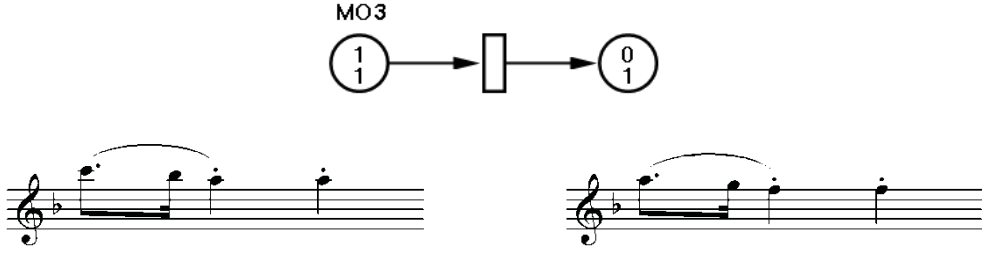
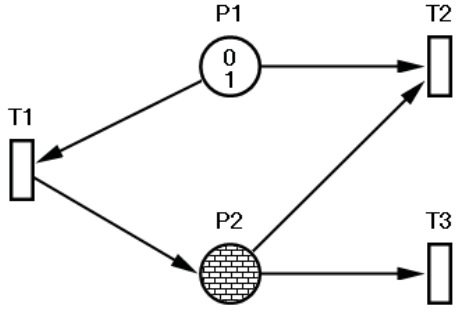


Fig. 4. Music Object 3 before (left) and after (right) the transposition

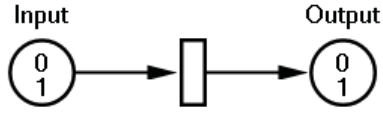
2.1 Extensions

In Music Petri Nets some basic extensions are considered. Since this formalism is used to represent the structure of music pieces, together with its hierarchies, the natural choice is to include the *refinement* as a simple morphism mechanism. With this extension, deeper music

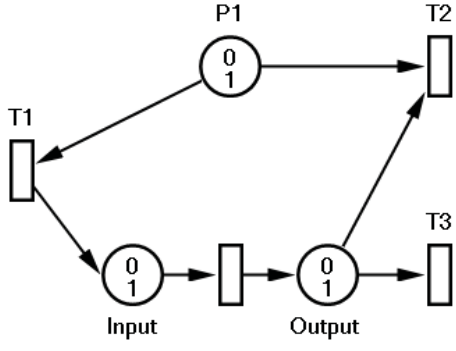
analyses can be integrated in subnets, permitting a better comprehension and decreasing the net complexity.
An example of refinement is presented in Figure 5. It must be noted that a node with a subnet must be of the same type of the subnet's input and output nodes, to achieve the expansion of the entire net.



a) The main net



b) The subnet P2



c) The expanded net

Fig. 5. An example of refinement

Another extension we have introduced is the *probabilistic weight* of arcs. This extension is used when fires of transitions are in conflict or in alternative, and is graphically represented by a numeric value over the arc, in square brackets. Normally, in non-deterministic

situations, which transitions will fire is randomly chosen. With the probabilistic weight, we can instead control this choice, even dynamically.
For example, let us consider the Petri Net in Figure 6 with 3 arcs: A_1 (probabilistic weight W_1 = 5), A_2 (W_2 = 10), and A_3 (W_3 = 300). If at a given time t_1 the choice is between all the three arcs, A_1 shall have a probability of 5/315 (1.6%) to fire, A_2 a probability of 10/315 (3.2%), and A_3 a probability of 300/315 (95.2%). At the time $t_2 > t_1$, let only A_1 and A_2 be enabled: their new probabilities will be 5/15 (33.3%) and 10/15 (66.7%) respectively.

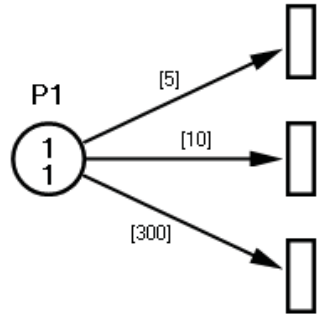


Fig. 6. The probabilistic weight extension

A particular situation occurs when an arc has a probabilistic weight equal to 0. In this case, the associated transition will fire only if there are no other alternative or conflicting arcs with greater probabilistic weight.

2.2 Music Petri nets applicability

At LIM[1], Petri Nets have been applied to music since 1982, and two paths were followed: music analysis and music creation. In other words, in some cases (Degli Antoni & Haus, 1983; Baratè et al., 2005) it has been investigated the possibility of describing causality in music processes through the formal approach of Petri Nets, while other studies focus on music creation ex-novo (Baratè et al., 2007).
It must be noted that different applications of Petri Nets to music analysis lead to apparently contradictory results. While Ravel's *Bolero* structure has been very well described in a convenient series of models (Haus & Rodriguez, 1993), some limitations of this approach have become evident when trying to describe, for example, the complexity of Stravinsky's *Rite of Spring* (De Matteis & Haus, 1996).
From the analytical perspective, excellent or poor results in representing music analysis through Petri Nets formalism mainly depend on three factors:

1. The intrinsic characteristics of the piece to be described. For instance, the music form known as *canon*, based on the literal repetition of the same music objects in different voices at different instants, can be represented in a very efficient and compact way with Petri Nets (see Figure 7).

[1] Laboratorio di Informatica Musicale, Università degli Studi di Milano.

2. The ability in confining those music objects which prove to be efficient from Petri Nets point of view. The concept of music object is deliberately vague and can include whole episodes of a music work, as well as atomic musical events. For this context, *segmentation* will be defined as the activity of isolating music objects and discovering their mutual relationships.
3. The degree of detail the analysis wants to reach. This statement can justify the contradictory results obtained when considering different music pieces. To illustrate this concept, we can consider for instance the test case presented at CMMR 2005 (Baratè et al., 2005), where the first movement of a sonata by W.A. Mozart is modelled. In Figure 8 is presented the very simple and compact net that describes the entire movement at the higher level of abstraction, while in Figure 9 the complexity of the transition in the recapitulation part of the piece is clear even without going into a detailed description of the Petri Net.

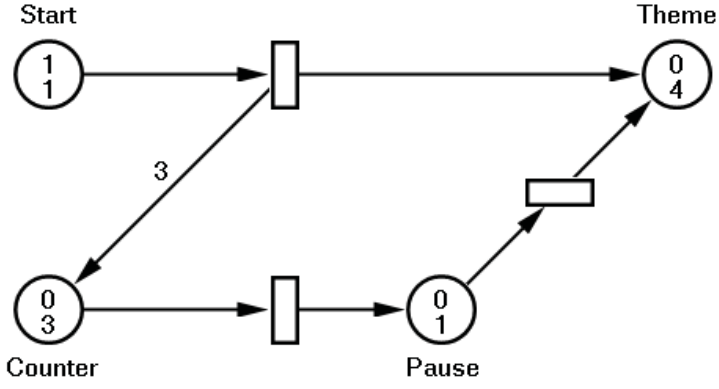


Fig. 7. The Music Petri Net of a canon

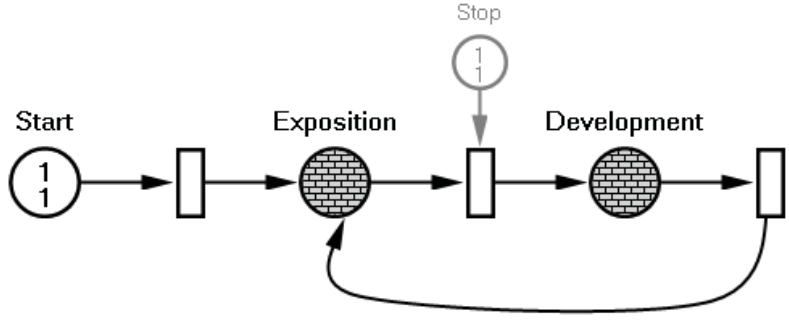


Fig. 8. The Music Petri Net of a sonata form

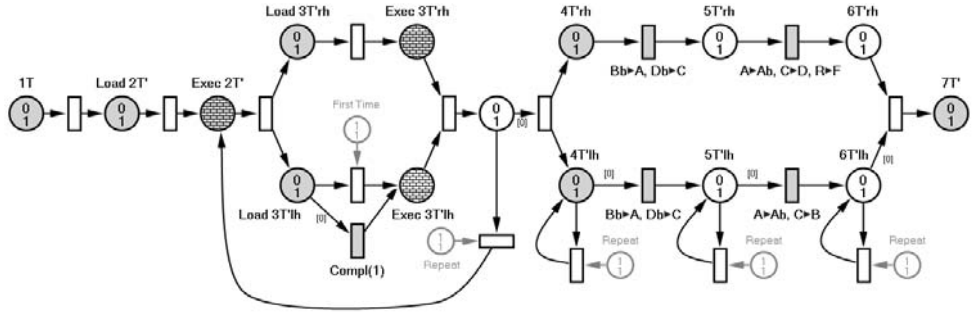


Fig. 9. The Music Petri Net of a part of the recapitulation in a sonata form

2.3. ScoreSynth

Music Petri Nets can be designed, developed and executed with an application named ScoreSynth (Figure 10). This application has an integrated environment to manage complex Petri Nets projects and to execute them in different ways:

- until no transitions are enabled;
- a step every "n" seconds;
- manual step by step;
- manual "object execution" by "object execution".

In ScoreSynth music objects associated to places are encoded in an XML format named MX.

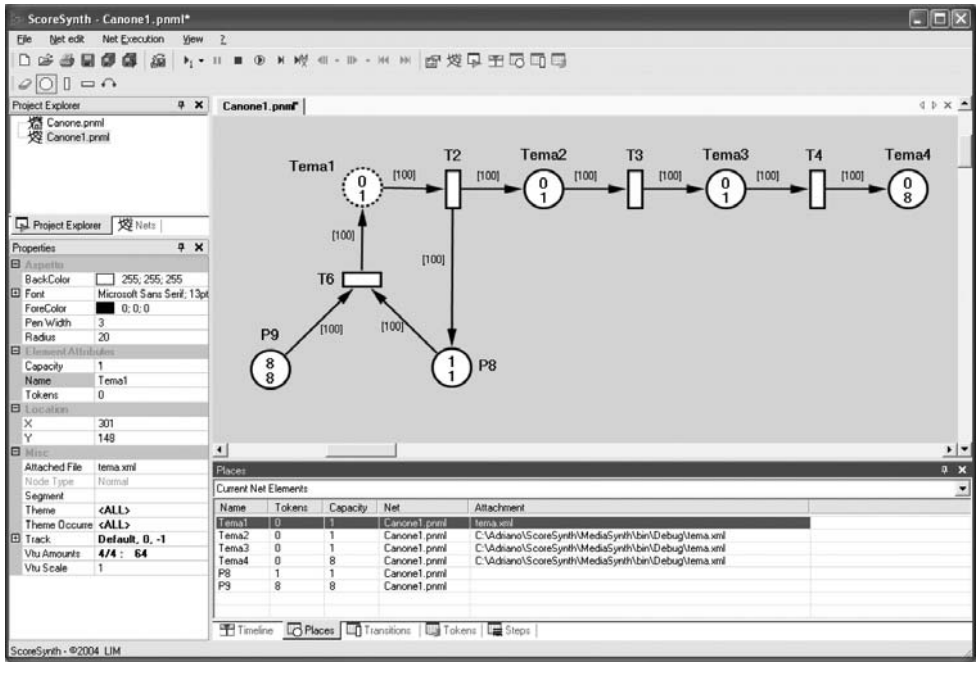


Fig. 10. The ScoreSynth interface

3. MX

MX is an XML dialect currently undergoing the IEEE standardization process (IEEE SA PAR1599). The main purpose of this format is having a comprehensive description of music (Haus & Longari, 2005). Even if specific encoding formats that represent peculiar music features, such as audio tracks or scores, are already commonly accepted and in use, they are not conceived to encode all this features together. On the contrary, we are interested in a comprehensive description of music, containing heterogeneous representations in a synchronized way.
In order to achieve a comprehensive description of music and complete synchronization among both homogeneous and heterogeneous representations of music contents, MX is based on two key concepts: an XML-based multi-layer structure and a space-time construct called spine. In the following sub-sections, we will define these concepts in detail.

3.1 Multi-layer structure

A comprehensive analysis of music richness and complexity highlights six different levels of music description: general, logical, structural, notational, performance, and audio layers (see Figure 11).

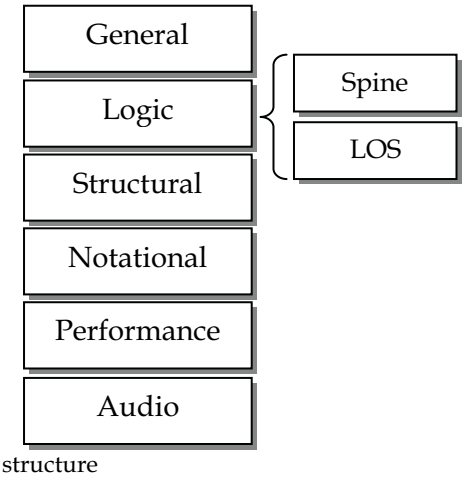


Fig. 11. MX multi-layer structure

General layer is mainly devoted to contain catalogue information about the encoded music piece. Logic layer contains information referenced by all other layers, and it is composed of two elements: the Spine, a sort of a "table of contents" used to mark music events in order to reference them from the other layers and the LOS (Logically Organized Symbols), that describes the score from a symbolic point of view (e.g. chords, rests). Structural layer contains compositional and musicological descriptions of the structure of the music piece (in this layer Music Petri Net links are allowed). Notational layer links visual instances of a music piece, such as digital images of the score. Performance layer links parameters of notes to be played and parameters of sounds to be created by a computer performance (e.g. MIDI and Csound). Finally, Audio layer describes audio information coming from recorded performances.

It must be noted that this approach allows MX to import commonly accepted formats aimed at music encoding, only by linking them in a particular layer, and then creating a mapping of the described events.

3.2 Spine

In order to synchronize the material described in all the MX layers, we introduced the concept of spine, a structure that relates time and spatial information. Spine is made of an ordered list of events, marked through a unique identifier to permit a reference from a particular layer. Each spine event can be described in different layers as well as in different instances within the same layer; e.g., in three different audio clips mapped in Audio layer.
Thanks to spine, MX achieves also a form of synchronization among layers (inter-layer synchronization) and among instances within a layer (intra-layer synchronization). Through such a mapping, it is possible to fix a point in a layer instance (e.g. Notational layer) and jump to the corresponding point in another one (e.g. Audio layer). This peculiarity was used in various applications to allow an evolved and integrated form of music enjoyment (Baggi et al., 2005; Baratè & Ludovico, 2007).

4. Music Petri nets and MX interaction

The adoption of the MX format in order to encode music objects in Music Petri Nets opens new possibilities to analysis and composition of music pieces. In a musicological perspective, an existing piece can be described by Petri Net models that can be linked together with other information in a single MX file. With specific applications, the analyst is able to have a global perspective at various levels of abstraction, described in different MX layers.
Another aspect that take advantages of this formalism is music creation. By using the MX format a composer could concentrate on the structure of the music piece he wants to obtain, without minding lower level material involved in the mixing process, such as the file formats of the linked objects. Thanks to MX, the final result automatically generates synchronisation of various kinds of music representation, permitting a new type of musical experience.

5. Acknowledgments

The author wants to acknowledge researchers and graduate students at LIM, and the members of the IEEE Standards Association WG on MX (PAR1599) for their cooperation and efforts. Special acknowledgments are due to: Denis Baggi, Goffredo Haus and Luca Andrea Ludovico for their invaluable work as working group chair, co-chair, and coordinator of the IEEE Standard Association WG on MX (PAR1599).

6. References

Baggi, D.; Baratè, A.; Haus, G.; Ludovico, L.A. (2006). Developing Intuition in Engineering Education: New Technology To Capture Structures in Music, *Proceedings of the 35th International IGIP Symposium, in cooperation with IEEE / ASEE / SEFI*, Tallinn, Estonia

Baratè, A.; Haus, G. & Ludovico, L.A. (2005). Music Analysis and Modeling through Petri Nets, In: *Computer Music Modeling and Retrieval*, R. Kronland-Martinet, T. Voiner, S. Ystad (Eds.), pp. 201-218, Springer Berlin Heidelberg, 3-540-34027-0, Berlin

Baratè, A.; Haus, G. & Ludovico, L.A. (2007). Petri Nets Applicability to Music Analysis and Composition, *Proceedings of the International Computer Music Conference '07 (ICMC 2007),* Holmen Island, Copenhagen, Denmark, 08-2007

Baratè, A.; Ludovico, L.A. (2007). An XML-based Synchronization of Audio and Graphical Representations of Music Scores, *Proceedings of the 8th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2007)*, Santorini, Greece

Degli Antoni, G. & Haus, G. (1983). Music and Causality, *Proceedings of 1982 International Computer Music Conference*, pp. 279-296, La Biennale, Venezia. Computer Music Association Publ., San Francisco

De Matteis, A. & Haus, G. (1996). Formalisation of Generative Structures within Stravinsky's "Rite of Spring". *Journal of New Music Research*, Vol. 25, No. 1, pp. 47-76

Haus, G. & Longari, M. (2005). A Multi-Layered Time-Based Music Description Approach based on XML. *Computer Music Journal*. MIT Press

Haus, G. & Rodriguez, A. (1993). Formal Music Representation; a Case Study: the Model of Ravel's Bolero by Petri Nets, In: *Music Processing*, G. Haus (Ed.), Computer Music and Digital Audio Series, pp. 165-232, A-R Editions, Madison