

PROGRAMMABLE LOGIC CONTROLLER

PROGRAMMABLE LOGIC CONTROLLER

EDITED BY
LUIZ AFFONSO GUEDES

Intech

Published by Intech

Intech

Olajnica 19/2, 32000 Vukovar, Croatia

Abstracting and non-profit use of the material is permitted with credit to the source. Statements and opinions expressed in the chapters are these of the individual contributors and not necessarily those of the editors or publisher. No responsibility is accepted for the accuracy of information contained in the published articles. Publisher assumes no responsibility liability for any damage or injury to persons or property arising out of the use of any materials, instructions, methods or ideas contained inside. After this work has been published by the Intech, authors have the right to republish it, in whole or part, in any publication of which they are an author or editor, and the make other personal use of the work.

© 2010 Intech

Free online edition of this book you can find under www.sciyo.com

Additional copies can be obtained from:

publication@sciyo.com

First published January 2010

Printed in India

Technical Editor: Teodora Smiljanic

Cover designed by Dino Smrekar

Programmable Logic Controller, Edited by Luiz Affonso Guedes

p. cm.

ISBN 978-953-7619-63-3

Preface

Despite the great technological advancement experienced in recent years, Programmable Logic Controllers (PLC) are still used in many applications from the real world and still play a central role in infrastructure of industrial automation. PLC operate in the factory-floor level and are responsible typically for implementing logical control, regulatory control strategies, such as PID and fuzzy-based algorithms, and safety logics. Usually PLC are interconnected with the supervision level through communication network, such as Ethernet networks, in order to work in an integrated form.

The first PLC were computers designed to specific proposal that worked with simple digital inputs and outputs, and their programming language were based on relay logic. Currently there is a wide range of PLC manufacturers that offers products to automate from domestic activities up to large scale industrial processes. Thus, there is a PLC for each type and class of application. The most powerful PLC are equipped with sophisticated hardware and software infrastructure. But the small PLC have configuration software with good features too.

Due to modern integrated automation concept, all components of the automation system must work interconnected through communication network and must be dotted of agile reconfiguration capability. Because PLC are the main equipments in several current automation solutions, there are a strong demand for standardized methodologies, technologies and software-based solutions to aid the various activities of the PLC programs development, such as modeling, validation, verification, test and automatic code generation. Other current demand is related with the difficulty in obtaining examples from the real world in order to explain how hard it is to develop application to PLC.

In this context, this book was written by professionals that work and research in automation area and it has two major objectives. The first objective is present some advances in methodologies and techniques for development of industrial programs based on PLC. The second objective is present some PLC-based real applications from various areas such as manufacturing system, robotics, power system, communication system, and education.

The book is organized in 10 chapters, where the first four are concerned with methodologies and techniques to develop PLC programs and the last six are PLC-based applications from the real world. We expect that the readers have basic knowledge of

industrial automation and PLC programming. On the one hand, since this book presents some recent advances in methodologies and techniques to help the development of PLC programs, we believe that it is useful for engineers, practitioners, graduate students and researchers who are related in the automation area. On the other hand, the chapters of applications can be especially useful for undergraduate student and engineers from several areas, such as computer, communication, electrical and mechanic engineering.

Editor

Luiz Affonso Guedes

*Federal University of Rio Grande do Norte,
Brazil*

Contents

Preface	V
1. Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic <i>Kwan Hee Han</i>	001
2. Practice of Industrial Control Logic Programming using Library Components <i>Oscar Ljungkrantz, Knut Åkesson and Martin Fabian</i>	017
3. Control and Plant Modeling for Manufacturing Systems using Basic Statecharts <i>Raimundo Moura and Luiz Affonso Guedes</i>	033
4. The Java based Programmable Logic Controller. New Techniques in Control and Supervision of a Flexible Manufacturing Cell. <i>Ramón Piedrafita and José Luis Villarroel</i>	051
5. Holonic Robot Control for Job Shop Assembly by Dynamic Simulation <i>Theodor Borangiu, Silviu Raileanu, Andrei Rosu and Mihai Parlea</i>	071
6. Centralized/Decentralized Fault Diagnosis of Event-Driven Systems based on Probabilistic Inference <i>Shinkichi Inagaki and Tatsuya Suzuki</i>	099
7. New Applications Using PLCs in Access Networks <i>Lamartine V. de Souza, João C. W. A. Costa and Carlos R. L. Francês</i>	121

8. Development of Customized Distribution Automation System (DAS) for Secure Fault Isolation in Low Voltage Distribution System 131
M. M. Ahmed, W.L. Soo, M. A. M. Hanafiah and M. R. A. Ghani
9. Computer Emulations to Support Training in Automation 151
Manuel E. Macías and Ernesto D. Guridi
10. PLC based Structure for Management and Control of Distributed Energy Production Units 161
Joao M. G. Figueiredo

Object-Oriented Modeling, Simulation and Automatic Generation of PLC Ladder Logic

Kwan Hee Han
Gyeongsang National University
Republic of Korea

1. Introduction

Most enterprises are struggling to change their existing business processes into agile, product- and customer-oriented structures to survive in the competitive and global business environment. Among their endeavor to overcome the obstacles, one of the frequently prescribed remedies for the problem of decreased productivity and declining quality is the automation of factories (Zhou & Venkatesh, 1999).

As the level of automation increases, material flows and process control methods of the shop floor become more complicated. Currently, programmable logic controllers (PLC) are mostly adopted as controllers of automated manufacturing systems (AMSs), and the control logic of PLC is usually programmed using a ladder diagram. More recently, manufacturing trends such as flexible manufacturing facilities and shorter product life cycles have led to a heightened demand for reconfigurable control systems. To cope with these challenges, a new effective and intuitive method for logic code design and generation is needed.

However, currently there are no widely adopted systematic logic code development methodologies to deal with PLC based control systems in the shop floor. So, the control logic design phase is usually omitted in current PLC programming development life cycle though it is essential to reduce logic errors in an earlier stage of automation projects before the implementation of control logic. Moreover, fast customer requirement changes requires flexibility of manufacturing system. To deal with these frequent configuration changes of modern manufacturing systems, it is required that logic code can be generated automatically from the design results without considering complicated control behavior.

To generate error-free ladder code, it is also essential to validate the designed control logic of an AMS in an effective way. Among many validation methods, computer simulation methods are widely used because mathematical formalisms have a problem of solution space explosion as the size of system increases. However, since current simulation methods have mainly focused on the overall performance evaluation of manufacturing systems such as factory layouts, resource utilization, and throughput time, they have limitations with regard to the modeling capabilities of detail logic for the input/output signal-level control of AMS. Therefore, current PLC ladder programming practices require a more integrated way to design, simulate, and generate the ladder control logic.

The main objective of this chapter is to propose an object-oriented (O-O) ladder logic development framework integrating design, validation and automatic generation of ladder

logic using extended UML (Unified Modeling Language). Proposed framework, as depicted in Figure 1, consists of three parts: first part deals with UML design of PLC-based control system. O-O design model consists of three models: functional model, structure model and interaction model. Second part is concerned with O-O simulation method for validating designed ladder control logic. By using the results of O-O design model, an O-O simulation model is constructed and is executed. During the execution of simulation model, factory automation (FA) engineers can evaluate the system performance and validate the PLC control logic simultaneously. Last part deals with automatic generation of ladder code from the validated design result. In order to show the applicability of proposed method, an UML-based tool for the design and generation of ladder code is also developed. Proposed framework facilitates the generation and modification of ladder code easily within a short time without considering complicated control behavior to deal with current trend of reconfigurable manufacturing systems.

The rest of the chapter is organized as follows. Section 2 reviews related works. Section 3 describes UML design of control logic. Section 4 deals with O-O simulation of designed PLC-based control system for validating the correctness of control logic. Section 5 describes the automatic generation and verification method of ladder logic. Finally, the last section summarizes results and suggests directions for future research.

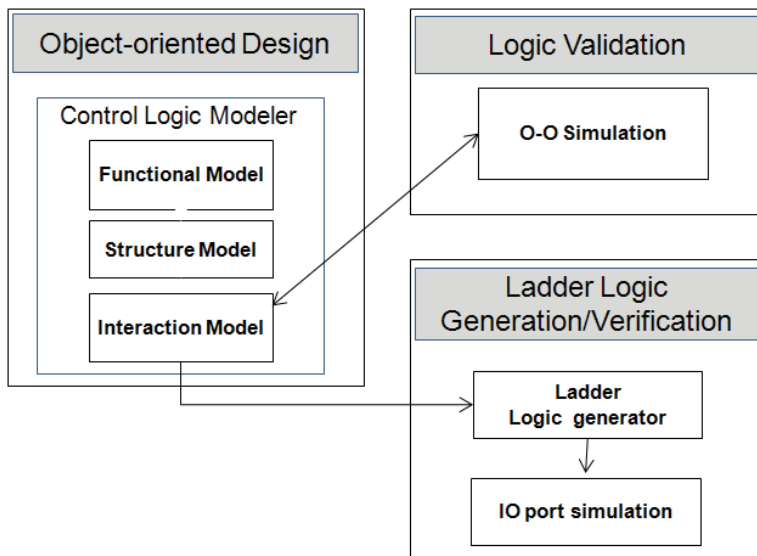


Fig. 1. Proposed object-oriented ladder logic development framework

2. Related works

In order to improve current PLC programming practices, significant efforts have been made in researches on object-oriented technologies in manufacturing systems. O-O modeling has been mainly used as a method for the analysis and design of software system. Recently, it is presented that O-O modeling is also appropriate for the real-time system design like an AMS as well as the business process modeling.

Several researches were made regarding the O-O modeling methods for the manufacturing system: Author of this chapter proposed AMS modeling framework called JR-Net (Job-Resource relation Net), which consists of a layout model, a functional model, and a control model for the O-O simulation of AMS (Choi *et al.* (1996), Park *et al.* (1997)). But, since this work placed emphasis on the supervisory control level rather than the device control level in the control model, it did not presented the modeling results of device level control. An O-O method for the design of automation system was proposed (Calvo *et al.*, 2002), but it only showed the static structure comprised of a class diagram and a use case diagram. An UML modeling of AMS and its transformation into PLC code was proposed (Young *et al.*, 2001), but it did not presented the method of PLC code generation. An UML modeling of flexible manufacturing system and its simulation implementation was proposed (Bruclerli & Diega, 2003), but it restricted the control level to the supervisory control level.

Among researches about design and validation tools for the PLC control logic, a simulation method integrating plant layout sub-model and control sub-model, and also a PLC code generation from simulation result was proposed (Spath & Osmers, 1996), but it omitted details of generation procedure. A procedure of control logic design was proposed by using IEC function block diagram (FBD) model, its transformation into Petri net, the validation of control logic using SIMULINK simulation system, and C code generation (Baresi *et al.*, 2000). But, it confined their modeling scope to simple control logic which can be represented by FBD. Author of this chapter developed O-O design tool based on the extension of UML and showed usefulness of O-O design and simulation approach to ladder logic development (Han & Park (2007a), Han *et al.* (2007b)).

In the area of automatic ladder logic generation method, there exist mainly three approaches as follows: First approach is Petri net-based (Peng & Zhou (2004), Lee *et al.* (2004), Frey & Minas (2001), Taholakian & Hales (1997)). Second approach is finite state machine-based (Jack (2007), Manesis & Akantziotis (2005), Sacha (2005), Liu & Darabi, (2002)). Last approach is flow chart-like-based (Jack (2007), Hajarnavis & Young (2005)). Among three approaches, first and second approaches have a state explosion problem when complexity of control logic increases.

The third approach is relatively easy to use by its sequential and intuitive nature to control logic programmers. However, the result of ladder code generated by the third approach proposed by Jack (2007) is different from the code directly written by FA engineers due to its automatic generation features. Therefore, it is not natural to FA engineers and revealed difficulties to understand the generated ladder code. Research about functionalities of Enterprise Controls commercial package of Rockwell Automation was presented, in which FA engineers design the ladder logic in the form of flow chart within Enterprise Controls, and ladder code is generated automatically (Hajarnavis & Young, 2005). However, it did not show how the ladder code is generated. Proposed generation method in this chapter belongs to the third category, in which ladder logic code is generated from the extended UML activity diagram which is a kind of flow chart.

3. Object-oriented design of control logic

The most typical features of O-O modeling techniques include the interaction of objects, hierarchical composition of objects, and the reuse of objects (Maffezzoni *et al.*, 1999). O-O design for ladder control logic is conducted based on system specifications such as drawings and problem descriptions. During the design phase, FA engineers develop three models for

describing the various perspectives of manufacturing systems: 1) a functional model for representing functional system requirements of AMS, 2) a structure model for representing the static structure and relationships of system components, and 3) an interaction model for representing the dynamic behavior of system components.

A functional model is constructed using an UML use case diagram in which each functional requirement is described as a use case. A use case diagram describes a top-level system view. A PLC as a plant controller is represented by a 'system' element, and input or output part of PLC such as a sensor, actuator, and operator is represented by an 'actor' element of a use case diagram. Since the UML stick man icon of 'actor' is not appropriate for representing the resource of AMS, new icons are introduced in a functional model using UML stereotype property. Therefore, in the extended UML use case diagram, as depicted in Figure 2, four types of actor (i.e., operator, actuator, sensor and MMI) are newly used instead of standard stickman symbol. PLC input parts such as sensor and operator are located at the left side of 'system' symbol, and PLC output parts such as actuator and MMI (Man Machine Interface) are located at the right side of 'system' symbol.

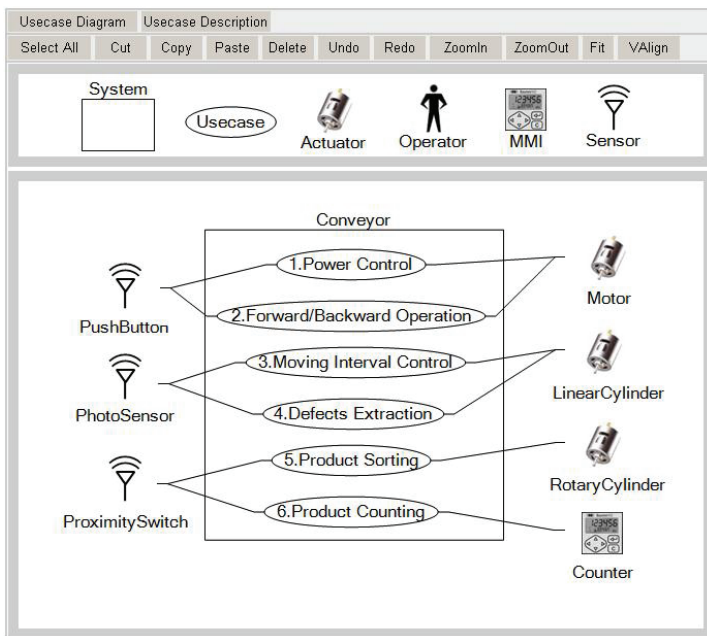


Fig. 2. Use case diagram of example prototype

The details of each use case are described in a use case description list. The use case description list includes the pre-/post-condition of a use case and interactions of a PLC with its actors such as sensors and actuators. For realizing a use case, related domain classes accomplish an allocated responsibility through the interactions among them. These related classes are identified in the structure model. And the system-level interactions in a use case description list are described in more detail in the interaction model.

Figure 2 and 3 shows a functional model for the example system in the form of use case diagram and use case description list. This example application prototype, as depicted in

Figure 4, is a kind of conveyor-based material handling system which identifies defective products according to their height, extracts defective products, and sorts good products according to their material property. It has 6 use cases for describing major functions from power control to product counting as depicted in Figure 2. Figure 3 shows the use case description of use case 4 (defects extraction) in Figure 2, and describes the high-level interactions between system (PLC) and its actors such as photo sensors and cylinders.

Usecase Description	
System : ConveyorSystem Name : Defects extration	
Scenario : Identifies defective products using 2 photo sensors. If defect product is identified, controller actuators extraction cylinder for the removal of product.	
Pre-condition : identification of defective products	
Post-condition : extraction of defective products	
Typical course or Events	
Actor	System
1. Product arrives 1.1 high level sensor is OFF and low level sensor is ON 1.2 high&low level sensors are all ON 1.3 high&low level sensor are all OFF 3. Extraction point sensor senses product 5. Extract defective product by forward stroke 6. Proximity switch senses good product	2. Identifies state of product 2.1 identifies good product 2.2/2.3 identifies defective product 4. Controller control cylinder according to the state of product 4.1 in case of defective products, sends forward stroke signal 4.2 initialize product status memory 7. Initialize product status memory

Fig. 3. Use case description list of example prototype

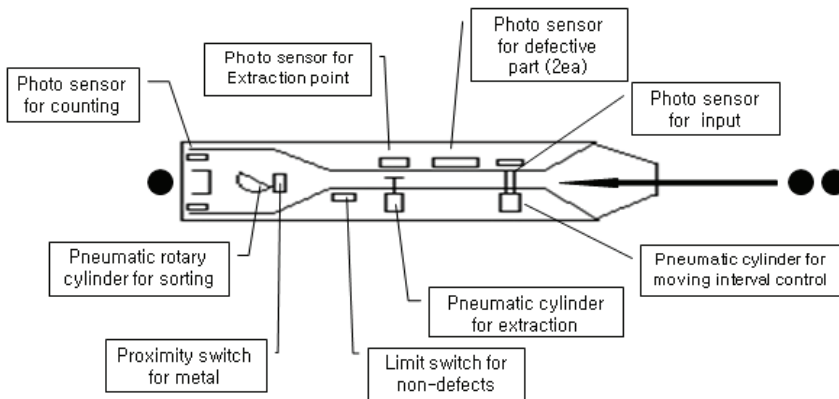


Fig. 4. Structure of example application prototype

A generic AMS is comprised of 4 parts: there is a 'plant' for manufacturing products. A plant is controlled by a 'controller' (PLC) which is managed by an 'operator' who monitors

plant through MMI. A 'work piece' flows through a plant. A plant is further decomposed into standard resource groups hierarchically.

Any standard resources can be classified using 3-level hierarchy of resource group-device group-standard device: A plant is composed of 'resource group' such as mechanical parts, sensor, actuator, and MMI. A resource group consists of 'device group'. For example, actuator resource group is composed of solenoid, relay, stepping motor, AC servo motor, and cylinder device group and so on. Sensor resource group is composed of photo sensor, proximity switch, rotary encoder, limit switch, ultrasonic sensor, counter, timer, and push button device group and so on. Finally, device group consists of 'standard devices' which can be acquired at the market.

To facilitate the modular design concept of modern AMS, the structure of AMS is modeled using an UML class diagram based on the proposed generic AMS structure. By referencing this generic AMS structure, FA engineers can derive the structure model of specific AMS reflecting special customer requirements easily. Figure 5 represents a static structure model of an example application prototype. Various kinds of device group class such as proximity switch and counter are inherited from generic resource group class such as sensor.

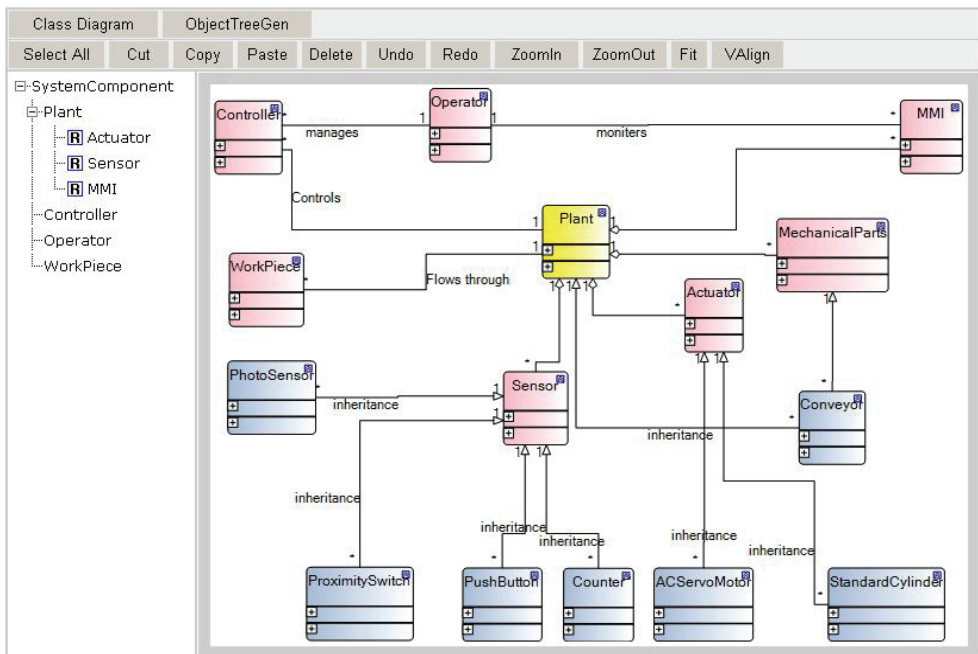


Fig. 5. Class diagram of example prototype

Since the real FA system is operated by the signal sending and receipt among manufacturing equipments such as PLC, sensors, and actuators, it is essential to describe the interactions of FA system components in detail for the robust design of device level control. This detail description of interactions is represented in the interaction model.

UML provides the activity diagram, state diagram, sequence diagram, and communication diagram as a modeling tool for dynamic system behaviors. Among these diagrams, the

activity diagram is most suitable for the control logic flow modeling because of following features: 1) it can describe the dynamic behaviors of plant with regard to device-level input/output events in sequential manner. 2) It can easily represent typical control logic flow routing types such as sequential, join, split, and iteration routing. The participating objects in the activity diagram are identified at the structure model.

In order to design and generate ladder logic, modification and extension of standard UML elements are required to reflect the specific features of ladder logic. First of all, it should be tested whether UML activity diagram is suitable for the description of control logic flow, especially for the ladder logic flow. The basic control flow at the ladder logic is sequence, split and join. Especially, three types of split and join control flow must be provided for ladder logic: OR-join, AND-join, AND-split. UML activity diagram can model basic control flows of ladder logic well.

Basically, ladder diagram is a combination of input contact, output coil and AND/OR/NOT logic. Since 'NOT' (normally closed) logic flow in the ladder logic cannot be represented directly in standard UML activity diagram, new two transition symbols for representing normally closed contact and negated coil are added as normal arcs with left-side vertical bar (called NOT-IN transition) or with right-side vertical bar (called NOT-OUT transition) as depicted in Figure 6. In the extended UML activity diagram, logic and time sequence flow from the top to the bottom of diagram.




base	extension		symbol
Transition	Normal Transition		
	Not Transition	IN	
		OUT	

Fig. 6. Extensions of transitions in AD

Figure 7 represents the interaction model for the identification and extraction of defective parts according to the height of products at the example application prototype. (Refer the use case number 4 in Figure 2 and use case description in Figure 3)

The control logic of Figure 7 for defects extraction is as follows: 1) $\text{High_Memory} := (\text{High_Sensor} + \text{High_Memory}) * \text{!Extract_Cyl}$, 2) $\text{Low_Memory} := (\text{Low_Sensor} + \text{Low_Memory}) * \text{!Extract_Cyl} * \text{!OK_LimitSwitch}$, 3) $\text{Extract_Cyl} := \{(\text{High_Memory} * \text{Low_Memory}) + (\text{!High_Memory} * \text{!Low_Memory})\} * \text{Extract_Sensor}$ where "!" means negation (NOT), "*" means conjunction (AND), and "+" means disjunction (OR).

4. O-O simulation for validating control logic

In this phase, O-O simulation model is constructed, and is executed for validating the designed control logic. When logic errors are found during the simulation execution, FA engineers correct logic errors and run the simulation model again. After validating control logic through simulation, FA engineers modify UML design model for reflecting the simulation result. In this way, the design-simulation cycle is done iteratively for error-free control logic.

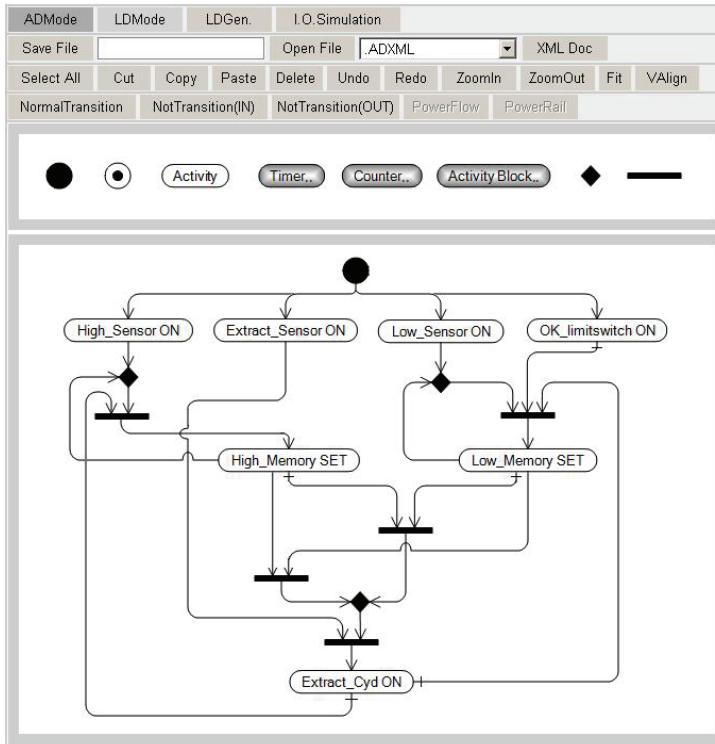


Fig. 7. Extended activity diagram for use case 4 in Figure 2 of example prototype

4.1 Construction of O-O simulation model

Based on the results of the O-O design model described in the Section 3, the O-O simulation model is constructed. The Unigraphics emPLANT software is used as an O-O simulation tool (Unigraphics, 2006).

First, for constructing an O-O simulation model, top-level functional requirements of automated manufacturing system are specified by using the use case diagram (Figure 2), and system-level interactions between the PLC and device actors (i.e., sensors and actuators) are identified by using a use case description list (Figure 3).

Second, AMS classes at the structure model (Figure 5) are mapped to emPLANT classes using the system hierarchy and association/inheritance relations among AMS classes identified in the class diagram. The mapping between generic AMS classes and emPLANT classes is summarized in Table 1.

Lastly, after determining the static system structure, control logic among system components is implemented for realizing each use case specified in the interaction model. The internal logic in the activity diagram is programmed in the simulation model by using SimTalk language of emPLANT software. For example, defects extraction of Figure 7 is executed by defect part identification and actuating extract cylinder. Detail control logic of this method is as follows: First, it inspects the product status according its height (a defective or good part). According to the inspection result, internal memories for high-level

and low-level detection are updated. If the product is defective and the sensor for extraction point is 'ON', the controller actuates an extraction cylinder. The SimTalk code of this logic is described in Table 2.

Generic AMS class		emPLANT class
Controller		Frame/ Method
Workpiece		Entity
Plant	Sensor	SingleProc/ Line-sensor
	Actuator	SingleProc
	Mechanical parts	Line/ SingleProc/ Transporter
	MMI	Frame/Method

Table 1. Mapping between generic AMS classes and O-O simulation elements

```
.Models.PLC.extract_cyd_ON
{
is
do
if ((high_Sensor=1 or high_Memory=1) and
(extract_Cyl=0))
then .models.conveyor_system.Plant.set_High_Memory;
end;
if ((low_Sensor=1 or low_Memory=1) and
(extract_Cyl=0) and (OK_Limit_Switch=0))
then .models.conveyor_system.Plant.set_Low_Memory;
end;
if ((high_Memory=1 and low_Memory=1) or
(high_Memory=0 and low_Memory=0)) and
(extract_Sensor=1)
then .models.conveyor_system.Plant.set_Extract_Cyl;
end;
end;
}
```

Table 2. Example of SimTalk simulation code for Figure 7

4.2 Execution of O-O simulation model

The main characteristics of the O-O model is the easiness of a top-down modeling approach because extended new classes which share common properties can be created by inheriting the pre-defined classes, and a system can be decomposed into sub-systems hierarchically.

The O-O simulation model of an example application prototype has two-level hierarchy. The high-level model for example prototype consists of a controller (PLC), a plant, a source of products, a storage of defective products, and a storage of good products (upper right part of Figure 8). Furthermore, this prototype can be abstracted to 2 components (i.e., a controller and a plant). The low-level model, which is a base model of simulation execution, decomposes the high-level model into more detailed elements such as sensors, actuators,

and MMI (lower right part of Figure 8). After constructing a simulation model and preparing an experimental frame, a simulation model can be executed in which product flows are animated through the conveyor line.

In parallel with the animation of products flow, the proposed O-O simulation model can show the animation of PLC operations in response to the various events about product flows (Left part of Figure 8). When the sensing of a product by various sensors is signaled to the input port of a PLC (input 'ON' signal), a PLC executes corresponding control logic and sends a signal to the output port of a PLC (output 'ON' signal). The output 'ON' signal is transmitted to the actuator, so the actuator is enabled.

As depicted in Figure 8, during the simulation execution, the ON/OFF animation of the PLC input/output ports is displayed in parallel with the product flows. Input ports are located at the left side of a PLC, and output ports are located at the right side of a PLC. The 'ON' signal of input/output ports is displayed by a red color at the screen display.

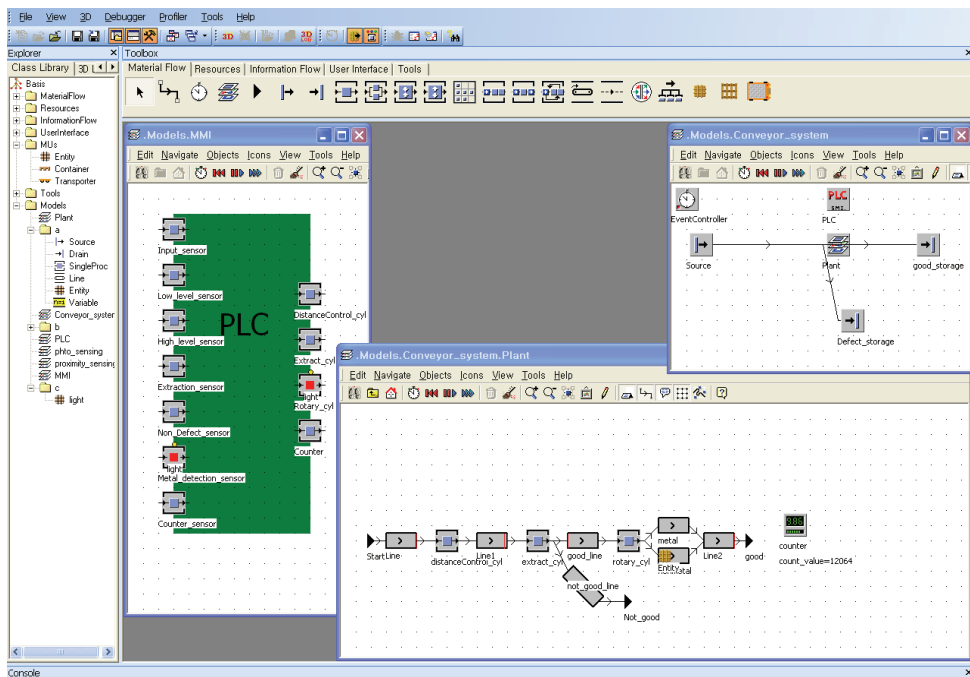


Fig. 8. O-O simulation model for example application prototype

Through the O-O simulation execution, PLC programmers can easily validate the internal logic of a PLC, and detect the logic errors at an earlier stage of the logic development by concurrent checking of product flows and PLC input/output port operations. Therefore, by adopting the proposed O-O simulation method, the validation of PLC control logic can be performed in parallel with the conventional performance evaluation.

5. Automatic generation of ladder code and its verification

The following two steps are conducted during the automatic generation phase: Firstly, ladder code is generated automatically using the interaction model result of design phase.

Secondly, generated ladder code is verified by input/output port-level simulation. In this phase, a software tool developed by research group including author is also used.

For the automatic generation of ladder logic, the mapping scheme of an UML activity diagram to a ladder diagram is established. IEC61131-3 standard ladder diagram have 5 major elements: contact, coil, power flow, power rail and function block (FB). Contact is further classified to normally open and normally closed contact. Coil is further classified to normal and negated coil. Power flow is further classified to vertical and horizontal power flow. Power rail is further classified to left and right power rail.

Elements of an activity diagram are classified to two types: an activity type and a transition type. Activity type is decomposed into start/stop activity, normal activity, special activity such as counter and timer, and block activity (Refer Figure 7). Transition type is decomposed into normal transition, NOT-IN transition for normally closed contact, NOT-OUT transition for negated coil, and logic flow transition. Logic flow transition is further decomposed into OR-join, AND-join and AND-split.

Figure 9 shows mapping scheme from an activity diagram to a ladder diagram. In order to store graphical activity diagrams and ladder diagrams in computer readable form, XML schema called AD-XML and LD-XML is devised for each diagram. In particular, LD-XML is an extension of PLCopen XML format (PLC Open, 2005).

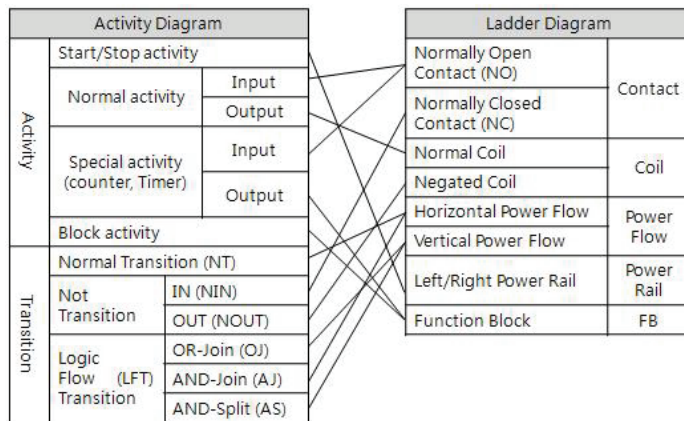


Fig. 9. Mapping scheme from AD to LD

After the activity diagram for specific control logic is stored in the form of AD-XML, AD-to-LD transformation procedure is conducted. Since basic ladder lung is a combination of input contact and output coil, an activity diagram is needed to be decomposed into several transformation units which having input(s) and output(s) corresponding to each ladder lung. This basic transformation unit is called IOU (Input Output Unit) which is a 1:1 exchangeable unit to ladder lung except start/stop activity. For example, the activity diagram depicted in Figure 10, which describes of power control logic (use case number 1 in Figure 2), has three IOUs. The control logic of Figure 10 is as follows: $\text{Conveyor_Motor} = (\text{PowerON_Button} + \text{Conveyor_Motor}) * \text{!PowerOFF_Button}$.

The transformation procedure is as follows: 1) After the creation of an activity diagram graphically, store it in the form of AD-XML. 2) Decompose an activity diagram into several input/output units called IOUs, and store it in the form of two-dimensional table called

IOU-Table. IOU-table has four columns named input activity, transition, output activity and IOU pattern type. Each row of IOU-Table becomes a part of ladder lung after the transformation process. 3) Determine the pattern type for each identified IOU. There are five IOU pattern types of activity diagram from the start/stop IOU type to the concatenation of logic flow transition IOU type. Generated IOU table for Figure 10 is shown at Table 3. 4) Finally, generate ladder lungs using IOU table and node connection information of AD-XML.

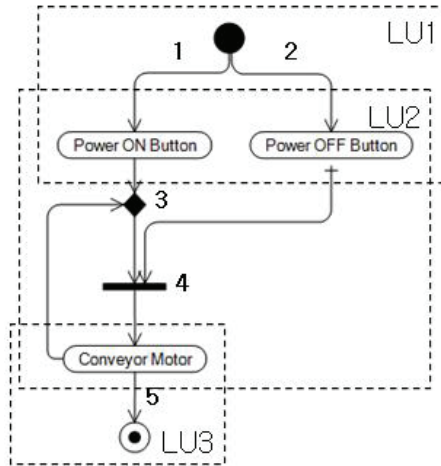


Fig. 10. IOU (Input Output Unit) decomposition

No.	Input Activity	Transition	Output Activity	Pattern
1	Start	1 : NT, 2 : NT	Power ON Button Power OFF Button	Type 1
2	Power ON Button Power OFF Button Conveyor Motor	3 : OJ, 4 : AJ	Conveyor Motor	Type 5
3	Conveyor Motor	5 : NT	Stop	Type 1

Table 3. IOU table for Figure 10 (use case 1-power control in Figure 2)

Figure 11 shows five IOU types and their corresponding LD patterns. IOU pattern type is classified to two types. One is simple type that is transformed to several basic ladder elements. The other is complex type that is a combination of simple types. Simple type is further classified to four types according to their corresponding lung structure: Type-1 (start/stop IOU), Type-2 (basic IOU), Type-3 (logic flow transition IOU: OR-join, AND-join, AND-split), and Type-4 (basic IOU with function block).

Since complex type is combination of several consecutive logic flow transitions, it has most sophisticated structure among 5 IOU types. Complex type is further classified to two types: Type 5-1 (join precedent) and Type 5-2 (split-precedent). Classification criteria is whether 'join' logic flow transition is precedent to other logic flow transitions or 'split' transition is precedent.

Lung Type		Description		AD pattern	LD pattern	
Simple Type	Type 1	start/stop IOU				
	Type 2	basic IOU				
	Type 3	3-1	logic flow transition IOU	OR-Join		
		3-2		AND-Join		
		3-3		AND-Split		
Type 4	basic IOU with Function Block					
Complex Type	Type 5	5-1	concatenation of Logic flow Transition (Join precedent)			
		5-2	concatenation of Logic flow transition (Split precedent)			

Fig. 11. Five IOU types

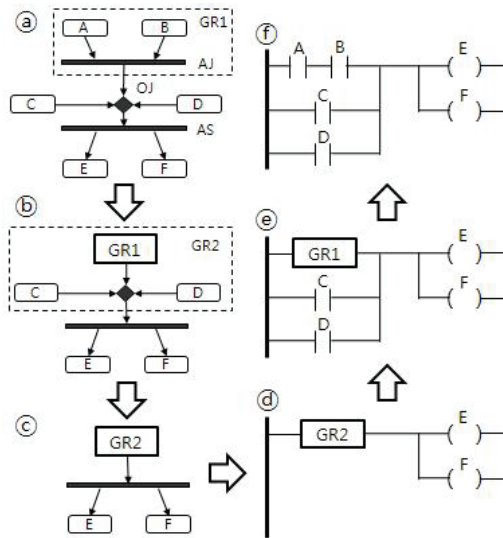


Fig. 12. Transformation procedure of join-precedent type 5-1

In order to transform the type-5 IOU to ladder pattern, hierarchical multi-step procedure is needed. The type-5 IOU is grouped hierarchically into several macro blocks for simplifying the consecutive control logic. A macro block is considered as a kind of block activity. Later, one macro block is transformed to one of five LD patterns. In other words, in order to simplify inputs for succeeding logic flow transition, firstly a macro block is built including precedent or succeeding logic flow transition. Later, a macro block is substituted by one of 5

ladder lung pattern. Fig. 12 shows the example of transformation procedure for the join-precedent type 5-1.

Ladder code is automatically generated based on the IOU table and node connection information of AD-XML. The generated ladder code is stored in the form of LD-XML, and is graphically displayed by reading LD-XML file as depicted in Figure 13. After ladder code is generated, it is necessary to verify the generated code. The simulation for code verification is conducted by input/output port level.

The ladder diagram in Figure 13 is generated from the control logic of activity diagram in Figure 7. As depicted in Figure 13, one can simulate the result of logic flow by closing or opening an input contact of specific lung, and monitoring the result of output coils and input contacts of other lungs.

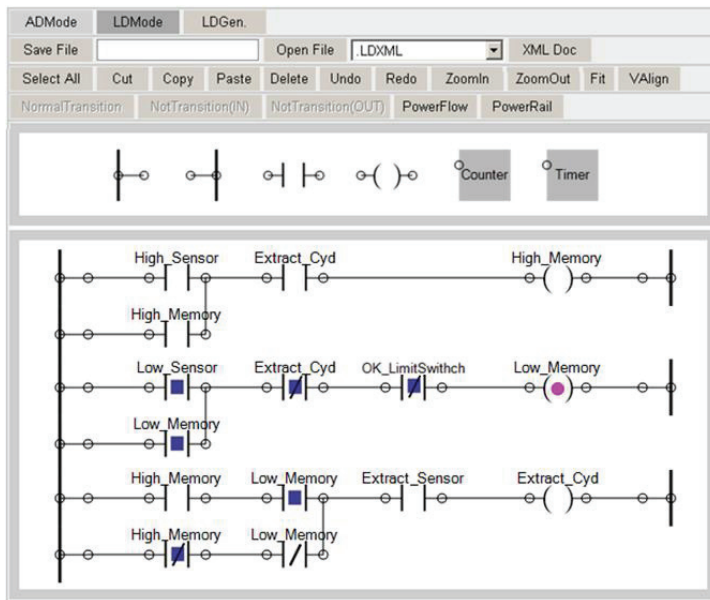


Fig. 13. Ladder code generation and port-level simulation of Figure 7

6. Conclusion

Currently, most enterprises do not adopt systematic development methodologies for ladder logic programming. As a result, ladder programs are error-prone and require time-consuming tasks to debug logic errors. In order to improve current PLC programming practices, this chapter proposes an integrated object-oriented ladder logic development framework in which control logic is designed, validated, generated automatically, and finally verified.

Proposed framework consists of three phases: First is the design phase. Second is the simulation phase. Third is the generation and verification phase. During the phase I, object-oriented design model is built, which consists of three sub-models: functional sub-model, structure sub-model and interaction sub-model. Based on the design result, O-O simulation model is constructed and executed for validating control logic during Phase II. After

correcting logic errors in Phase II, two steps are conducted during the phase III. Firstly, ladder code is generated automatically using the validated interaction model of design phase. Secondly, generated ladder code is verified by input/output port simulation. A framework in this chapter facilitates the generation and modification of ladder code easily within a short time without considering complicated control behavior to deal with current trend of reconfigurable manufacturing systems. In addition, this framework serves as a helpful guide for systematic ladder code development life cycle. As a future research, reverse transformation method from a ladder diagram to an activity diagram is needed for the accumulation of ladder logic design documents since design documents of control logic are not well prepared and stored in the shop floor.

7. References

- Baresi L., Mauri M., Monti A., and Pezze M. (2000). PLCTools: design, formal validation, and code generation for programmable controllers, *Proceedings of 2000 IEEE Conference on Systems, Man and Cybernetics*, Nashville, USA
- Bruccoleri M., and Diega S. N. (2003). An object-oriented approach for flexible manufacturing control systems analysis and design using the unified modeling language, *International Journal of Flexible Manufacturing System*, Vol.15, No.3, pp.195-216
- Calvo I., Marcos M., Orive D., and Sarachaga I. (2002). Using object-oriented technologies in factory automation, *Proceedings of 2002 IECON Conference*, pp.2892-2897, Sevilla, Spain
- Choi B.K., Han K.H., Park T.Y., (1996). Object-oriented graphical modeling of FMSs, *International Journal of Flexible Manufacturing System*, Vol.8, No.2, pp.159-182
- Frey G. and Minas M. (2001). Internet-Based Development of logic controllers using signal interpreted Petri nets and IEC 61131, *Proceedings of the SCI 2001*, Vol.3, pp.297-302, Orlando, FL, USA
- Hajarnavis V. and Young K. (2005). A comparison of sequential function charts and object modeling with PLC Programming, *Proceedings of American Control Conference*, pp.2034-2039
- Han K. H. and Park J. W. (2007a). Development of object-oriented modeling tool for the design of industrial control logic, *Proceedings of the 5th International Conference on SERA*, pp.353-358, Busan, Korea
- Han K. H. , Park J. W. and Choi Y. (2007b). Object-oriented modeling and simulation for the validation of industrial control logic, *Proceedings of the 37th international conference on CIE*, pp. 2377-2384, Alexandria, Egypt
- Jack H. (2007). Automating manufacturing systems with PLCs. <http://claymore.engineer.gvsu.edu/~jackh/books.html>
- Lee G. B., Zandong H. and Lee J. S. (2004). Automatic generation of ladder diagram with control Petri net, *Journal of Intelligent Manufacturing*, Vol.15, No.2, pp.245-252
- Liu J. and Darabi H. (2002). Ladder Logic Implementation of Ramadge-Wonham supervisory controller, *Proceedings of Sixth International Workshop on Discrete Event Systems*, pp.383-389
- Maffezzoni C., Ferrarini L., and Carpanzano E. (1999). Object-oriented models for advanced automation engineering, *Control Engineering Practice*, Vol.7, No.8, pp.957-968

- Manesis S. and Akantziotis K. (2005). Automated synthesis of ladder automation circuits based on state diagrams, *Advances in Engineering Software*, Vol.36, No.4, pp.225-233
- Park T.Y., Han K.H., Choi B.K., (1997). An object-oriented modeling framework for automated manufacturing systems, *International Journal of Computer Integrated Manufacturing*, Vol.10, No.5, pp.324-343.
- Peng S. S. and Zhou M. C. (2004). Ladder diagram and Petri net based discrete event control design methods, *IEEE Transactions on Systems, Man and Cybernetics-Part C.*, Vol.34, No.4, pp.523-531
- PLC Open (2005). XML formats for IEC 61131-3, <http://www.plcopen.org>
- Sacha K. (2005). Automatic code generation for PLC controllers, *LNCS 3688*, pp.303-316
- Spath D., and Osmer U. (1996). Virtual reality- an approach to improve the generation of fault free software for programmable logic controllers, *Proceedings of IEEE International Conference on ECCS*, pp.43-46, Montreal, Canada
- Taholakan A. and Hales W. M. M. (1997). PN <-> PLC: a Methodology for designing, simulating and coding PLC based control systems using Petri nets, *International Journal of Production Research*, Vol.35, No.6, pp.1743-1762
- Unigraphics (2006), emPlant, www.ugs.com/products/tecnomatix/plant_design/em_plant.shtml.
- Young K. W., Piggan R., and Rachitrangan P. (2001). An object-oriented approach to an agile manufacturing control system design, *International Journal of Advanced Manufacturing Technology*, Vol.17, No.11, pp.850-859
- Zhou M. C. and Venkatesh K (1999). Modeling, simulation and control of flexible manufacturing systems, *World scientific publishing*, Farrer Road, Singapore

Practice of Industrial Control Logic Programming using Library Components

Oscar Ljungkrantz, Knut Åkesson and Martin Fabian
*Department of Signals and Systems
Chalmers University of Technology
Sweden*

1. Introduction

This chapter discusses *Programmable Logic Controller (PLC)* programming practice, particularly the use of library components, in the automotive industry. A study of program structure and use of library components at two European car manufacturers is presented. The main purpose of the study is to provide understanding of current PLC programming in industry.

PLCs are commonly used in mass-production for instance to coordinate robots and machines. The life-cycles of many mass-produced products, including automotive products, have decreased significantly during the last years, due to changing market demands and increased competition. This has put new requirements on PLC programs, which must be easily modifiable and quickly made fully operational, to decrease down-time and ramp-up-time of the production system (Mehrabi et al., 2000).

PLCs are traditionally manually programmed in any of the languages of the *IEC 61131-3* standard (IEC, 2003; Lewis, 1998). Especially *Ladder Diagrams (LDs)*, derived from the time when physical relays were used to control the machines, are common (Johnson, 2002). To gain reusability and modifiability, PLC code can be encapsulated and reused as *function blocks (FBs)*. Nonetheless, the traditional PLC programs tend to be difficult to modify and extend and not flexible enough to meet the new requirements (Lewis, 2001).

A solution to the problems might be to use frameworks that facilitate the development of flexible and operational control programs. Hence, many researchers have developed new frameworks and tools to develop or automatically generate PLC code to meet the new requirements. Overview of such frameworks can be seen in (Lee et al., 2006; Ljungkrantz & Åkesson, 2007). In spite of the potential benefits of these academic frameworks, they have not been reported to be used in full scale industrial projects. One obstacle is that the generated code in practice often has to be modified by hand and integrated with working code already existing in industry.

For any code generating framework to be industrially successful, it certainly has to fulfil the requirements of industry. Moreover, successful integration of the generated code with already existing code requires understanding of PLC programming practice. This chapter aims at providing this knowledge. The chapter focuses on FB usage since reusing FBs created at the manufacturing companies is a promising approach for performing the code

integration. Most results and findings are based on a study performed 2007 at two Swedish car factories, which is reported in (Ljungkrantz & Åkesson, 2007) and is restated with some additional comments and findings in Section 2-5 of this chapter. A comparable study was performed at Lamb Technion in USA (Lucas & Tilbury, 2003). That study was however focused on the development process and not on library components. Furthermore, only LD programming was used in that study, while this chapter presents the use of other languages and programming constructs as well.

This chapter describes three major observations:

- The PLC programs in the studied companies were written mainly in Ladder Diagrams and Sequential Function Charts. These programs frequently reused function blocks.
- The PLC programs handled, besides automatic control, also safety and supervision, human machine interface, product data, communication etc. The code for automatic control was a minor part of the total code.
- Although the function blocks were frequently reused, their behaviours were only informally described.

To improve the efficiency and reliability when reusing FBs, we think it's crucial that the FBs are unambiguously specified and verified. The end of this chapter therefore shows how FBs can be formally specified and then verified using *model checking*. Model checking means to automatically check whether or not a model fulfils a specification (Clarke et al., 2000). Thus, model checking complements the traditional methods of testing and simulation. FBs can be augmented with formal specifications to form components we call *Reusable Automation Components (RACs)* (Ljungkrantz et al., 2008), which can be verified using model checking. An example FB is specified and verified as a RAC; an error is detected, the implementation is corrected and the final RAC is successfully verified. This shows the potential of using formal methods in function block development.

1.1 Chapter organization

This chapter is organized as follows: Section 2 describes the scope and methods of the study. In Section 3, the control program development at the studied companies is explained. In Section 4, the most frequently used library components are presented and discussed and in Section 5 a classification and statistics of the library components FBs, are presented. Section 6 discusses formal specification and verification of FBs and applies these techniques on an example FB. Conclusions are given in Section 7.

2. Study of control code and library components

The program structure and the library components in PLCs used at two Swedish car companies were studied in (Ljungkrantz and Åkesson, 2007). Mainly the code used in the car body assembly factories located in Sweden was investigated, since the PLCs in those factories control many robots, conveyors and other machines and have quite standardized layout. Other PLC programs at the two companies may be different from those studied. Still, "Company 1" and "Company 2" will from now on be used to refer to the respective studied factories and "the studied companies" will be used when referring to both. The investigation was performed by 1) manually reading the code in the PLC program development tool, 2) discussing with PLC engineers and programmers at the studied companies and studying a master thesis performed at the companies (Bergqvist and Öberg,

2007) and 3) writing a program that searches through PLC code and libraries and extracts FB usage statistics. At the time of the study, the studied companies used the same PLC program development tool, see Section 3.2.

The PLC code investigated was structured as different *projects*, each representing the code that runs on one PLC. In many cases one PLC controls one manufacturing cell, but in some cases two or more PLCs are used for one cell. Normally a cell is divided into several stations and a PLC often controls more than one station. For a fair comparison between the studied companies nine similar projects were chosen at each company: two *underbody* projects, three *respot* projects, one *side line* project, one *framing* project and two *transportation* projects. In the underbody cells, robots weld/bolt parts together to form the floor of a car. In the respot line the car floor or body are transported between the cells by a conveyor system and in each cell robots perform extra welding/bolting to increase the strength and to add extra parts. In the side line cells the sides of the car are built. In the framing cells the car body is built by welding together the car floor with sides etc. In the transportation cells conveyors, lifts etc. transport the car floor or body.

The projects and the libraries were exported to text files. The developed program reads those files and detects all instances of each FB. It detects both FB instances that are used directly in the projects and FB instances that are used indirectly. FBs are considered to be used indirectly if they are used inside an FB, which has instances directly used in a project or in turn is used indirectly. The program presents usage statistics of the FBs.

All used FBs were also classified into nine different categories, see Section 5. The program presents the number of instances and proportion of each category.

3. Control program development

This section describes the development of PLC programs at the studied companies, by describing the development actors, programming environment and general program structure.

3.1 Important actors of the development process

The PLC programs are usually developed by firms contracted by the studied companies. These firms program in a certain structure by following guidelines at the studied companies. Company 1 has a written specification for control programming and a standard project to start from. Company 2 has a stricter standard and structure of the code for the developers to follow. Both of the studied companies also provide libraries with components to reuse. The consultants may add components into the library, but these components are reviewed by the studied companies. At the time of the study, Company 1 had one person responsible for the library but a team of people that could review the code. Most of the components had no documentation apart from comments within the components. Hence, to understand the behaviour of a component, its internal code and comments had to be examined. At Company 2 a single person reviewed and also documented all library components. The documentation at Company 2 was done using pictures and natural language and was connected to the library components (as help files in the PLC program development tool). In addition to these internally developed and maintained libraries, suppliers of certain equipment also provide libraries with components to use with that

equipment. Finally, the supplier of the PLC hardware and the development tool also provided a number of libraries to use.

3.2 Development environment and programming languages

At the time of the study, PLCs from the same vendor were used at both of the studied companies. The development tool used to program these PLCs supports programming in the IEC 61131 standard (IEC, 2003). Hence the programs can be written in five languages: *Sequential Function Chart (SFC)*, *LD*, *Function Block Diagram*, *Instruction List* and *Structured Text* (to be precise, SFC is not considered a language in the standard, merely a graphical technique or program structure). The standard defines components called *POUs*, Program Organisation Units, to be reused and stored into component libraries. POU's can be of three different types: *functions*, *FBs* and *programs*. Functions may have many inputs but only one output. They have no memory and are typically used for mathematical operations. FBs allow an algorithm or set of actions to be applied to a given set of data, including inputs and internal variables, to produce a new set of output data. The behaviour of the FBs can be implemented in any of the five IEC 61131 languages and FB instances can be used in code written in any of the five IEC 61131 languages. Note that although IEC 61131 allows it, the used PLC program development tool did not permit the behaviour of FBs to be implemented using SFC.

3.3 General program structure

Both Company 1's and Company 2's projects consisted of several programs. Typically most programs were written in LD, one in Instruction List and up to two programs per station in SFC. A main sequence SFC of each station normally controlled the main order in which the operations of robots, clamps, transportation systems etc. should be performed. At Company 1 the robots also allocated resources (machines or virtual zones), before they for instance started welding, to avoid collision. This was done by having a separate LD program for each robot that handled interlocks and allocation of resources needed by the robot. At Company 2 this resource allocation was not needed since the sequence itself guaranteed that no collisions and variations occurred.

At Company 2 only nine types of programs were identified: two general programs ("Always" and "PLC_General", both LDs), one program for the Profibus communication (Instruction List), four programs for each station X (two SFCs, "StnX_Auto" and "StnX_Homerun", and two LDs, "StnX_Manual" and "StnX_General") and finally two built-in supervision programs provided by the PLC supplier. Company 1's projects were split into more types of programs: five to ten general programs (for communication, finishing the line, indication, communication with the safety PLC etc., all LDs), one program for the Profibus communication (Instruction List), many programs for each station X and robot Y (up to two SFCs, "SXMain" and "SXHomeRun", and many LDs, for instance "SXMovement", "SXTransport", "SXSumMemories", "SXBodyId", "SXAAlarms", "SXRobotsY" and "SXIndications"). While Company 2's projects for instance had alarm handling code inside the actions of the SFC, at Company 1 the actions of the SFC were mainly used to set variables that in turn were used in the LD programs. For simple transportation cells neither Company 1 nor Company 2 used SFC.

Although most of the programs were implemented in LD many FB instances were used and called from the LD code. Some programs almost resembled Function Block Diagrams. The behaviour of almost all FBs was implemented in LD.

The studied companies had few levels of hierarchy in the sense that FBs, apart from basic FBs, seldom were used inside other FBs. Company 2 argued that blocks inside other blocks make it hard to read and understand the code. Both of the studied companies put emphasis on the importance of having code that can be understood and used in trouble-shooting by the operators; this affected both the structure and naming of the code and the comments. Ideally, the alarms and indications of a PLC project are sufficient for the operators to solve problems but since this is not always the case, the code must be readable by the operators.

4. Frequent library components

At both of the studied companies programs were reused indirectly by starting from copies of standard projects or programs. Only the built-in supervision programs at Company 2 were reused as is. Two built-in basic libraries provided with the PLC program development tool consisted of both functions and FBs for mathematical operations, bit-manipulating etc. The rest of the libraries almost exclusively consisted of FBs. Therefore the investigations focused on FB reuse.

To illustrate the use of FBs at the studied companies, the five most frequently used FBs, according to the projects investigated, are briefly described here. Then some of the FBs are further described in an example of controlling two parallel clamps and the approaches chosen at Company 1 and Company 2 are compared. The most frequent FBs are presented in Table 1. They represented approximately 50 % of the total number of FB instances.

	Company 1		Company 2	
	FB Name	FB Instances Dir. / Indir.	FB Name	FB Instances Dir. / Indir.
1	FB_Event	893 / 19	OUT_SV x	380 / 13
2	FB_Move	301 / 8	ManAuto	78 / 0
3	FB_Alarm_Clamps	269 / 0	Valve_ctrl x	74 / 4
4	FB_Event_Clamps	266 / 0	CycleTime	70 / 0
5	FB_AllocateZon	226 / 0	EM_Status	54 / 0

Table 1. Most used FBs at the studied companies in the investigated PLCs.

4.1 Company 1

FB_Event

FB_Event uses a counter to assure that its binary output signal is held high for a minimum time, when its binary input goes high. The purpose is to assure that signals sent to other systems keep their values long enough to be detected. It should be used for all signals sent via TCP/IP. Furthermore, in the study FB_Event was used at almost all binary status signals sent to actuators, supervision and HMI systems.

FB_Move

FB_Move, see Figure 1, controls the movement of actuators like clamps, fixation pins and lifts. It can be used for moving the actuator both backwards and forwards in either automatic or manual mode. If for instance a forward movement in automatic mode is ordered by signal *AutoFwd*, some conditions are checked and if those are fulfilled the output signal *OutputFwd* goes high. At the same time a timer is started and when the timer has reached the value of *TimeValue* the *TimeOutFwd* output goes high.

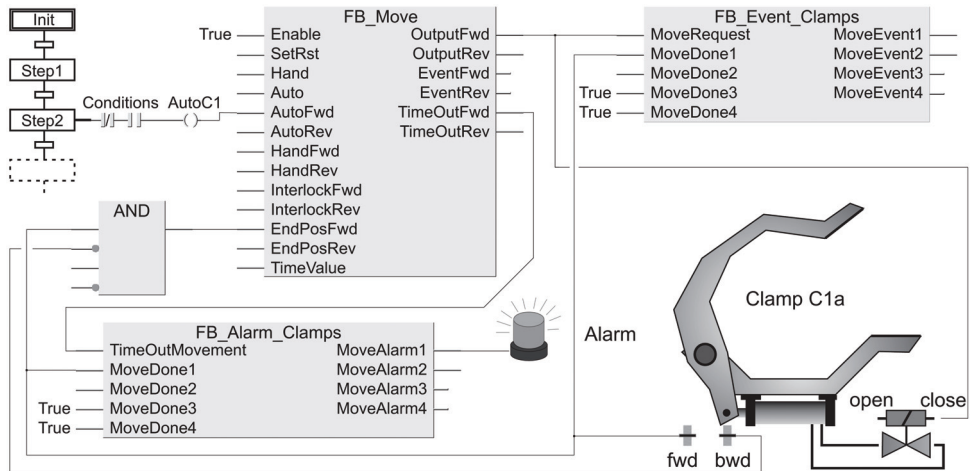


Fig. 1. The principle of controlling two parallel clamps at Company 1. To increase readability, the components for one of the clamps and for backward movement are omitted.

FB_Alarm_Clamps

FB_Alarm_Clamps is used to send an alarm if the movement of any of up to four parallel clamps is not performed within a specified time, see Figure 1. The input signal *TimeOutMovement* is activated by an external timer and when this signal goes high FB_Alarm_Clamps sends alarms for all clamps that have not yet reached the end position.

FB_Event_Clamps

FB_Event_Clamps FB has the same purpose as FB_Event but can be used for up to four parallel clamps. It has five input signals: a move request for the whole clamp group and four signals telling whether the connected clamps have reached their end positions or not, see Figure 1. Inside FB_Event_Clamps the four signals each goes through an FB_Event.

FB_AllocateZon

FB_AllocateZon is used to handle interlocks between a robot and a machine or between different robots, to avoid collision. A robot FB sends a unique number, representing the resource that the robot wants to use, to the FB_AllocateZon. The FB_AllocateZon checks that the conditions are met and when so it sends back the number representing the resource.

4.2 Company 2

OUT_SV_x

OUT_SV and OUT_SV_x ($x = 2, \dots, 6$ is the number of parallel movements to supervise) are FBs included in the built-in supervision library provided with the PLC program development tool, see OUT_SV2 in Figure 2. These FBs are used to supervise movements by checking that the movement has stopped within a specified time after the *Run* signal is given. Otherwise alarms are given for the movements that are not finished. The output signal *Out* shall be connected to the component that shall be moved.

ManAuto

The ManAuto FB was in the study used once for every station at Company 2. The FB handles the choice for running in automatic or manual mode and has input signals for the desired mode and for emergency stop, acknowledge signals for Profibus communication etc.

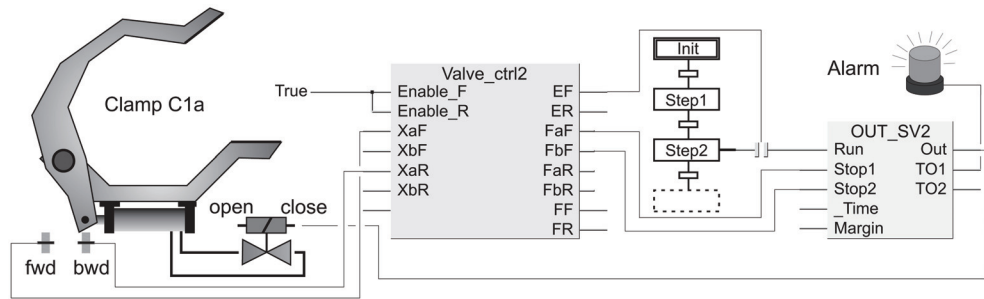


Fig. 2. The principle of controlling two parallel clamps at Company 2. To increase readability the components for one of the clamps and for backward movement are omitted.

If all conditions are fulfilled, the desired mode is chosen. The *Au* and *Ma* output signals were used as conditions for conveyors, robots and actuators, either as a logic condition for a specific movement or as a condition for a whole program. The latter was used for the programs that handled only automatic or manual control of a station. When the ManAuto output FBX.Ma was *true* it activated a *task*, see (IEC, 2003), so that program StnX_Manual ran. When instead FBX.Au was *true* it activated a built-in function SFC_CTRL so that the SFC StnX_Auto ran.

Valve_ctrlx

Valve_ctrl and Valve_ctrlx ($x = 2, \dots, 13$ is the number of parallel actuators) are FBs to control one or many actuators connected to one valve, see Valve_ctrl2 in Figure 2. The end position sensors, backwards and forwards, for each of the connected actuators are input signals to the block. If the *Enable_F* input signal is *true* the forward output *EF* is set if all actuators are in backward position. The FB also has output signals for each of the actuators stating if the actuator is in forward and not backward position, and vice versa.

CycleTime

The FB CycleTime calculates the cycle time for a station by increasing a counter each second when the station is not paused, and resetting every new cycle.

EM_Status

EM_Status identifies and sends an error message from an electric monorail conveyor.

4.3 Example and comparison

In this section the main control approach at the studied companies will be explained using a simple example, in which many of the above FBs will be used. The task is to close a clamp group, consisting of two clamps that are moved in parallel via one pneumatic valve connected to the cylinders of both clamps. Each clamp has sensors in both end positions.

The components for controlling the clamps at Company 1 are shown in Figure 1. The movement is started when the main sequence of the station, implemented as an SFC, is in the position where the clamps should be closed. If some basic conditions are satisfied (for instance that the station cycle has not already been performed in manual mode) the FB_Move is told to start the movement of the clamps. If the station is in Auto, the clamp group is not already closed etc., FB_Move starts a timer and sends a signal to the valve to close the clamp group. This signal is also sent to the FB_Event_Clamps. If any of the two clamps is not closed within the maximum time allowed for the clamp group, the FB_Alarm_Clamps sends an alarm for that clamp. The AND operator is used to assure that

both clamps are closed and not open. In the figure all FBs for one clamp and for closing the clamp are shown. Components for the second clamp (C1b) should be added in the same way, components for opening the clamp should also be added in a similar way. The FBs FB_Event_Clamps and FB_Alarm_Clamps can be used for up to four clamps. As seen, fewer clamps can be controlled by setting the unused input sensor signals to *true* and letting the corresponding output signals be unconnected. Four of the five most used FBs at Company 1 in the study are used in the example (FB_Event is used inside FB_Event_Clamps). The signal out of the AND operator is a typical signal that can be used as an interlock for the fifth most used FB, FB_AllocateZon, for instance guaranteeing that the clamps are closed before the robot welds the part held by the clamps.

The components for controlling the clamps at Company 2 are shown in Figure 2. When the clamp group is open this is known by the Valve_ctrl2, since all four end position sensors are connected to this FB, and the EF (enable forward) output is high and the FaF and FbF outputs are low since the clamps are not in closed position. The real movement is started first when the auto sequence of the station, implemented as SFC also at Company 2, is in the position where the clamps should be closed. Now the OUT_SV2 FB tells the clamps to close. If a clamp is not in forward position before the time *_Time* has passed, an alarm is raised.

The approaches at the two companies in the study were quite similar, as exemplified above, letting an SFC start the movement and reusing common FBs, with LD to describe the logic. Nevertheless, there were also small but interesting differences. The function of the AND operator in the Company 1 example was instead included in the Valve_ctrlx FB using LD, at Company 2. At Company 2 different FBs were needed when different numbers of clamps were to be controlled, as indicated by the number succeeding the FB name (for instance Valve_ctrl2 and OUT_SV2). This means that Company 2, in this case, had to keep and maintain more FBs in the library, but on the other hand did not have to set unconnected inputs to *true* or *false*. The OUT_SVx FBs that were used at Company 2 are very similar to Company 1's FB_Alarm_Clamps, but are included in the built-in supervision library provided with the PLC program development tool. A benefit of using OUT_SVx FBs is that they can be given a teach mode in which the supervision program detects the actual time before the stop signals are detected and updates the *_Time* parameter with the measured time plus the *Margin*, given in %. Finally, the FB_Event was very common at Company 1 but not used at Company 2.

5. Classification and statistics of function blocks

In 2007, Company 1 had recently started classifying their in-house libraries into function based categories. Company 2 had chosen a more equipment-based classification. To be able to compare the libraries we divided the FBs into nine categories. All FBs in frequent libraries and all used FBs have been classified. The categories are listed below.

- *Robot Control*: Control of, and resource allocation for, robots.
- *Machine Control*: Control of other machines than robots, e.g. actuators and conveyors.
- *HMI*: FBs for indication, mode-choice and manual control. Interaction with the operators.
- *Safety and Supervision*: FBs for alarm handling, communication with the safety PLC and automatic safety operations like emergency stop.

- *Product and Production Data*: FBs for communicating with identification systems like barcodes and RFID, and for controlling the production by for instance choosing next product type.
- *Statistics*: Data collection and calculations for analysis, for instance cycle time, product counters and mean time between failures.
- *Ethernet & Profibus Communication*: Communication protocols, drivers etc. for Profibus and Ethernet.
- *General Functions*: FBs like timers, clock settings and bit-manipulating, maintained by the studied companies.
- *Basic*: The FBs in the two built-in libraries *Manufacturer_Lib* and *Standard_Lib*, provided with the PLC program development tool. FBs for basic mathematical operations, bit-manipulating etc.

At Company 1, 249 FBs have been classified and 141 of those were used in the investigated projects, including basic FBs. At Company 2, 200 FBs have been classified and 80 were used in the investigated projects, including basic FBs.

In the investigated projects Company 2 had 1338 FB instances and Company 1 had 4514 FB instances. Ignoring the Basic FBs they still used 1115 and 4128 FB instances respectively. At Company 1 *FB_Event* and *FB_Event_Clamps* accounted for almost 30% of all FB instances. Besides, they were not used at Company 2 and used in many different circumstances at Company 1, so placing them in a single category would be inaccurate. Hence, they have been excluded when counting how many FB instances that are used within each category. Even with *FB_Event* and *FB_Event_Clamps* excluded Company 1 used 2950 FBs which is significantly more than Company 2's 1115. Although it was the intention to choose similar projects from Company 1 and Company 2, a reason for the difference may be more extensive PLC projects at Company 1. The difference might also be due to different structure and usage of FBs within the projects, at Company 1 and Company 2. This explanation is indicated by the clamp control example depicted in Figure 1 and 2, showing three directly used FB instances and one indirectly used FB instance (*FB_Event* inside *FB_Event_Clamps*) at Company 1 but only two FB instances at Company 2. The FB instances divided into the different categories can be seen in Table 2.

Category	Used FB instances [%]	
	Company 1	Company 2
Robot Control	22	6
Machine Control	17	15
HMI	19	7
Safety and Supervision	25	50
Product and Production Data	6	5
Statistics	5	8
Ethernet & Profibus Com.	4	7
General Functions	3	2

Table 2. Percentages of used FB instances divided into different categories.

The FB instances do not represent the complete code, neither do they directly correspond to the work done by the developers. For instance the Ethernet & Profibus communication instances were quite few in the study but each FB was often complex. Still, the FB instances

do represent a rough estimation of how the PLC code was divided. For instance the code handling HMI, safety, supervision, communication etc. undoubtedly represents a great part of the code. In (Lucas and Tilbury, 2003; Richardsson and Fabian, 2006) it is reported that according to their experience at Lamb Technion and Volvo Car Corporation respectively, the part of the code representing automatic control is about 10 % of the total. However, no data supporting this was shown in the two papers. In the investigation reported here the code for automatic control was *part of* the categories robot control and machine control, accounting for in total 39 % at Company 1 and 21 % at Company 2. For instance, Company 1's FB_Move, classified as machine control, was directly called from the SFC handling the automatic control. At Company 2 the EF output of Valve_ctrlx, classified as machine control, was directly used in the action logic of the SFC for automatic control. Nevertheless, some FBs classified as machine or robot control, especially at Company 1, handle low level control of the machines and robots and should not be considered code for the automatic control itself, rather help FBs for the code which handles the operation order for automatic control. Therefore we can not claim that the code representing automatic control is exactly 10 % of the total, but it is indeed fair to state that: *the code for automatic control is a minor part of the total code.*

It is also interesting to compare the category distribution at the two companies. Robot control is a greater part at Company 1 than at Company 2, which could be explained by the fact that Company 1 assumes that the operations can be executed in different orders and therefore uses zones to allocate resources. Company 1 also uses FBs for lamp indication (HMI) more frequently. The proportion of FB instances for alarm handling (safety and supervision) is significantly greater at Company 2. This can be explained by considering an SFC with parallel branches. At Company 2, the alarm handling FBs were included in the SFC and thus two instances of the involved FB existed in an SFC with two parallel branches and so on. At Company 1, the SFC branches set variables that in turn were used in separate LD programs, containing only one instance of the involved FB. In this particular case, the choice at Company 1 resulted in more compact code, while the code at Company 2 may be considered easier to read.

6. Formal specification and verification of function blocks

With the above findings as starting point, it is the authors' belief that the code reuse can be made more efficient and less error prone. Efficient code reuse indeed requires components with known behaviour. This can be achieved by developing clear and unambiguous *specifications* and by verifying that the specifications are fulfilled by the *implementation* (the code). The specification can be seen as an abstraction of the implementation, capturing important properties.

As explained in Section 3.1, most FBs at Company 1 had no external documentation. The internal comments of the FBs are in principle insufficient as specification, since these comments are too strongly connected to the implementation (possibly violating the principle of abstraction) and reading the comments require access to the implementation (violating the principle of information hiding, see (Parnas, 1972)). The external documentation of the FBs at Company 2 does not have these disadvantages. Nonetheless, being based on natural language, both the comments of Company 1 and the documentation of Company 2 might be ambiguous and not suitable as a basis for verification. In particular, this natural language documentation is not suitable when using *formal verification*.

Formal verification uses math-based models and algorithms to perform the verification and thus requires a formal and unambiguous specification. *Model checking* is an important set of formal verification methods that can perform the verification automatically and produce counterexamples if the specification is not fulfilled (Clarke et al., 2000). Model checking is promising in FB development, since compared to common field testing, model checking can be performed earlier in the development process. Model checking has also advantages compared to simulation, since in many situations it is too time consuming to simulate and test all different scenarios in which a component can be used. Model checking however typically performs exhaustive search of the models.

Model checking PLC code can be done using many different methods and tools, see (Bérard et al., 2001; Frey and Litz, 2000). The *Reusable Automation Component (RAC)* method developed by the authors of this chapter is tailored for specifying and verifying PLC program components, such as FBs (Ljungkrantz et al., 2008). The RAC specification structure and language is intended to be understandable by PLC program engineers without prior knowledge on formal languages. A RAC prototype tool has also been developed with which the RACs can be specified and then automatically translated into inputs to the model checking tool Cadence SMV (McMillan, 1993, 1999). The RAC method and tool is used here to demonstrate the usefulness of formal specification and verification in FB development. Next, the basics of the RAC are explained, followed by an example component that controls actuators similarly to the examples seen in Figure 1 and 2. This example component is not very complicated but still shows the advantages of using formal verification. Formal specification and verification of the more complex component *FB_Move* used by Company 1 can be seen in (Ljungkrantz et al., 2008).

6.1 Reusable Automation Components (RACs)

The RACs were introduced in (Ljungkrantz et al., 2008). A RAC has an *interface* that includes inputs and outputs and a *body* that includes the implementation and internal variables. The main difference compared to FBs is that the RAC interface includes a formal specification. As help when developing and structuring the specification, five types of properties can be used, briefly described below:

- *Operation preconditions* are requirements that the user of the component must satisfy in order to obtain certain functionality, expressed by the *operation behaviours*.
- *Operation behaviours* are requirements, ensured by the developer of the component, that must be fulfilled when all operation preconditions are satisfied.
- *Exception conditions* are prioritized inputs or combinations of inputs that lead to exceptions. When an exception condition is *true* none of the operation behaviours can be guaranteed. Instead the exception condition must always guarantee certain behaviour, which must be described as *exception behaviours*.
- *Exception behaviours* are requirements, ensured by the developer of the component regardless of the operation preconditions. Each exception behaviour includes one or more exception conditions.
- *Invariants* are requirements, ensured by the developer of the component regardless of operation preconditions.

The operation preconditions and operation behaviours are grouped as *operation specification* and the exception conditions and exception behaviours are grouped as *exception specification*.

The *specification language* is based on IEC 61131-3 (all four languages but not SFC) and *Linear Temporal Logic (LTL)*, see for instance (Clarke et al., 2000). The reason for basing the specification language on IEC 61131-3 is that most PLC engineers are familiar with the IEC 61131-3 languages but might not know other programming or specification languages. Augmenting the language with constructs for LTL is done to express relations over time. Temporal logic contains constructs to reason about the order in time without explicitly mentioning time; for instance it can state that something will *always* or *eventually* be *true*. LTL is a type of temporal logic that suits the input-output based relations of FBs well and is also supported by model checking tools. The specification language contains spelled out versions of the temporal operators but also short-hand notations for some basic constructs, like rising and falling edges of variables. For instance the rising edge of a boolean variable v can be expressed as $v_risingEdge$ which is equivalent to $(NOT\ v_previous) \ \&\ v$, using the Structured Text based variant of the specification language.

6.2 Example

As an example, the development of a RAC *Control_BinaryActuator*, implemented as a function block in LD, will be demonstrated. The RAC should control a binary actuator and should signal alarms if the movements are not performed within a maximum time. Hence this RAC will contain most parts of the components *Valve_ctrl* and *OUT_SV* from Company 2, see Section 4.2 and many parts of the components *FB_Move* and *FB_Alarm_Clamps*, excluding interlocks and mode handling, from Company 1, see Section 4.1.

Assume that the interface of the example component has already been determined. The inputs and outputs of *Control_BinaryActuator* can be seen in Figure 3, which also shows how the component can be used to control a cylinder. When the *Move* input is *true*, the actuator will move forwards by setting *ActuatorFwds* to *true* if the *DesiredState* is "Forward" and move backwards by setting *ActuatorBwds* to *true* if *DesiredState* is "Backward". Move must be held *true* throughout the complete movement. When the movement has been performed, as indicated by the sensor inputs *SensorFwd* and *SensorBwd*, the *State* output will be set to the new state. The component also has checks to see if the actuator performs accurately and outputs alarm signals if not. If the movement is not performed within the maximum time allowed, *MaxMoveTime*, the corresponding alarm, *TimeOutActFwds* or *TimeOutActBwds*, will be set *true*. The *AlarmUnauthMove* alarm is set if the actuator moves when it is not supposed to. Finally, the alarms can be reset by the user, by setting *ResetAlarms* to *true*.

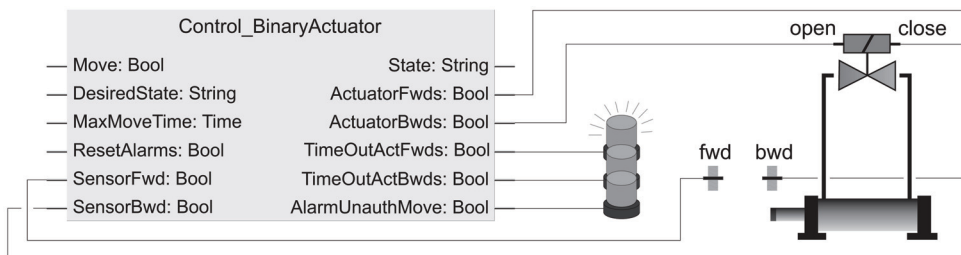


Fig. 3. The inputs and outputs of the *Control_BinaryActuator* RAC.

The specification of *Control_BinaryActuator* can be seen in Figure 4, using the Structured Text based variant of the specification language.


```

Operation specification
Operation preconditions
OpPre1 := ALWAYS( (MaxMoveTime > 0) &
                  ((DesiredState = Forward) OR (DesiredState = Backward)) );
OpPre2 := ALWAYS( Move & Move_previous ->
                  (DesiredState = DesiredState_previous) );

Operation behaviours
MoveOrAlarm := ALWAYS( (ALWAYS Move) -> EVENTUALLY((State = DesiredState) OR
                                                    TimeOutActFwds OR TimeOutActBwds) );

Exception Specification
Exception conditions
Reset := ResetAlarms;

Exception behaviours
ResetBhvr := ALWAYS( ResetAlarms ->
                    (NOT TimeOutActFwds & NOT TimeOutActBwds & NOT AlarmUnauthMove) );

Invariants
NotIllegalMove := NEVER( ActuatorFwds & ActuatorBwds );
Stop := ALWAYS( (NOT Move) -> (NOT ActuatorFwds & NOT ActuatorBwds) );

```

Fig. 4. Specification of the *Control_BinaryActuator* RAC.

The first operation precondition states the allowed input values for *MaxMoveTime* and *DesiredState*. The second operation precondition states that the user must not change the direction of the movement while moving. The operation behaviour *MoveOrAlarm* summarizes the main functionality of the RAC by stating that when the user of the RAC is trying to move the actuator, the actuator will eventually reach the desired state or an alarm is raised. More operation behaviours could be added, for instance to specify under what circumstances the operating outputs *ActuatorFwds* and *ActuatorBwds* are actually *true*, but for brevity only *MoveOrAlarm* is shown. The exception condition *Reset* states that none of the operation behaviours can be guaranteed if the *ResetAlarms* input is *true*. The corresponding exception behaviour *ResetBhvr* declares that the *ResetAlarms* input will always reset all three alarms. The invariant *NotIllegalMove* states that the RAC will never try to move the actuator in both directions simultaneously. Finally, *Stop* declares that the outputs that move the actuator will never be *true* when *Move* is *false*. Note that *NOT Move* could as well have been specified as an exception condition, depending on how “normal” operation of the component is viewed. If so, *Stop* would have been specified as an exception behaviour instead.

Implementation and Verification

A rather straightforward attempt of implementing the example component can be seen in Figure 5. The implementation makes use of the standard functions *AND*, *EQ* (tests equality) and *MOVE_E* and of the function block *TON*. *TON* is a standard timer that sets the output *Q* to *true* if *IN* is *true* at least as long as the time *PT*. The function *MOVE_E* that is used in the position control at the top of Figure 5, copies the string on the *IN* input to the output to which *State* is connected, when the *EN* input is *true*. The positive (P) and negative (N) transition-sensing contacts are used to detect rising and falling edges of the signals, respectively.

The RAC can now be formally verified to check whether the implementation of Figure 5 fulfils the specification of Figure 4 or not. The RAC can be translated into inputs to Cadence

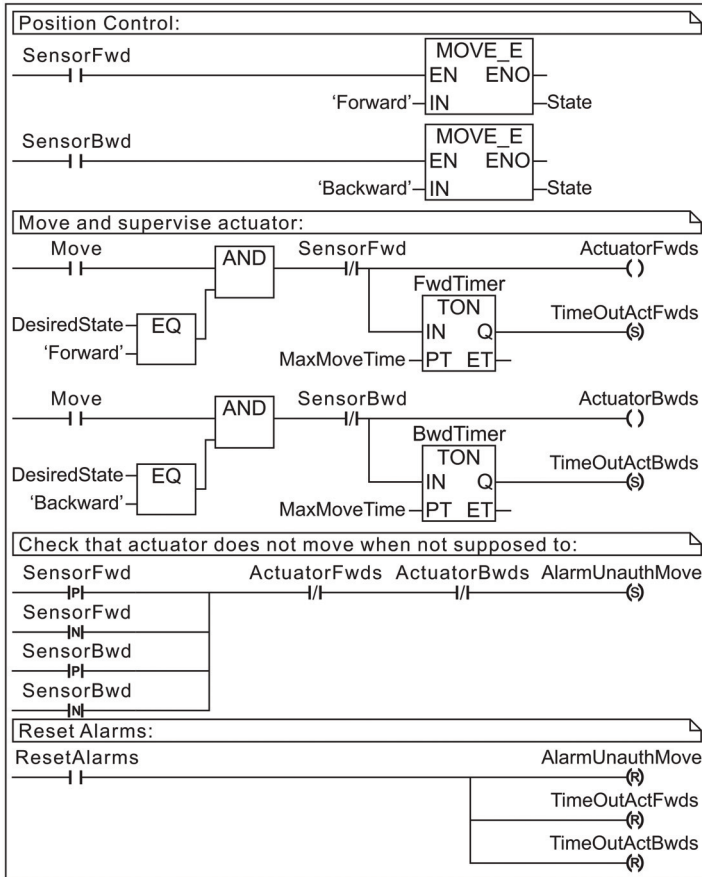


Fig. 5. An implementation approach of the *Control_BinaryActuator* RAC in Ladder Diagrams using standard functions and the timer FB TON.

SMV by the RAC prototype development tool, and then Cadence SMV can be used to perform the verification. Doing this, the result is that the RAC is *not valid*, that is the specification is *not fulfilled* by the implementation. Both invariants are fulfilled, but not the operation behaviour *MoveOrAlarm*. SMV gives a counterexample to why the operation behaviour is not fulfilled to help understand and solve the problem. If the actuator should be moved forward but the sensors are broken so that both sensor inputs are *true* at the same time, the actuator will not be moved and unfortunately the *TimeOutActFwds* will not be set. The *FwdTimer* will not be started since the *SensorFwd* is already *true*, but the state will be reported as *Backward* (from the second *MOVE_E* function) and hence *MoveOrAlarm* is not fulfilled. The RAC could certainly be made valid by adding a precondition saying that the actuators and sensors may never be broken, but a much better alternative is to change the implementation so the alarms will actually work when the sensors are broken.

To solve the problem, two internal variables *InFwdPosition* and *InBwdPosition* are used that are *true* only when *SensorFwd* and not *SensorBwd* are *true* and vice versa. Those internal variables are used as conditions to start the timers, as shown in Figure 6. Using this

implementation the complete specification is fulfilled and the RAC is valid. Even for such a small and elementary component as `Control_BinaryActuator`, the error of the first implementation attempt might be hard to foresee. By studying the counterexample of the model checking tool though, the error can be easily solved. This demonstrates the potential of using formal specification and verification in the FB development process.

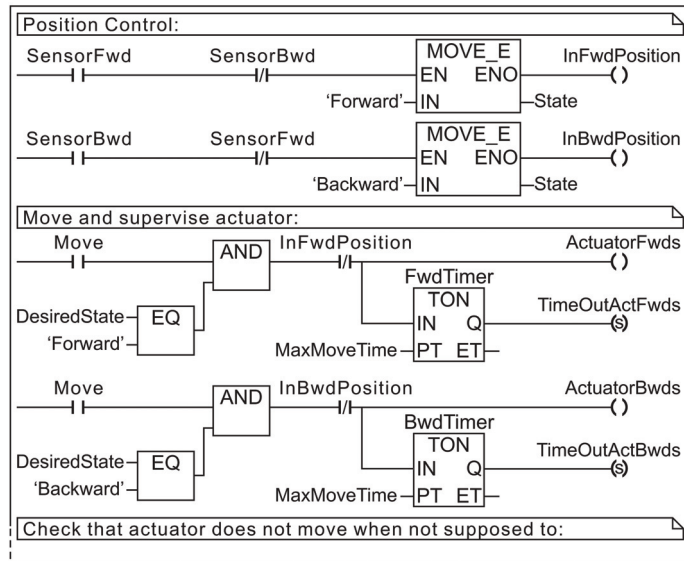


Fig. 6. A valid implementation of the `Control_BinaryActuator` RAC in Ladder Diagrams. The part not shown is exactly the same as in Figure 5.

7. Conclusions

In this chapter a study of PLC programming and use of library components at two Swedish car manufacturers is presented. Both companies used several programs for each PLC, implemented mainly in LD and SFC. These programs included lots of instances of reusable function blocks, FBs. Some of the most frequent FBs were used for automatic control of actuators and conveyors but in total only a minor part of the used FB instances was for automatic control; the majority was for HMI, safety, supervision, production data, communication etc. This is important to consider when developing or modifying frameworks for control program generation, to cope with the new requirements of flexible manufacturing systems. Integrating industrial FBs with new frameworks for generating control sequences is an interesting direction for future research.

It is also interesting to consider that although the FBs were frequently reused, their behaviours were only informally specified. In our opinion the FB reuse can be made more efficient by also using tools and methods for formal specification and verification. This is demonstrated by an example component, in which an error of the first implementation attempt is discovered and solved.

For formal specifications to be used in industry it is important that the development of relevant specifications is not too troublesome or time consuming. We therefore currently research into developing guidelines for formal specification of PLC program components.

8. Acknowledgment

This research is financed by the ProViking research programme. Thanks also to all concerned staff at the studied companies for sharing their knowledge and code. Thanks to Isak Öberg and Olof Bergqvist for performing an interesting master thesis.

9. References

- Olof Bergqvist and Isak Öberg. PLC function block survey of Swedish automotive industry. Master's thesis, Dept. Signals and Systems, Chalmers Univ. Technol., Göteborg, Sweden, 2007.
- B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification – Model-Checking Techniques and Tools*. Springer, 2001.
- Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.
- Georg Frey and Lothar Litz. Formal methods in PLC programming. In *Proc. Int. Conf. Syst., Man, Cybern.*, pages 2431–2436, Nashville, TN, USA, 2000.
- IEC. Programmable Controllers—Part 3: Programming languages. International standard IEC 61131-3. International Electrotechnical Commission, second edition, 2003.
- Dick Johnson. Nano devices lead assault on traditional PLC applications. *Control Engineering*, 49(8):43–44, 2002.
- Seungjoo Lee, Mark Adam Ang, and Jason Lee. Automatic generation of logic control. Technical report, Ford Motor Co., Univ. of Michigan and Loughborough Univ., 2006.
- Robert W. Lewis. Programming industrial control systems using IEC 1131-3 Revised edition. The Institution of Electrical Engineers, 1998.
- Robert W. Lewis. *Modelling Control Systems Using IEC 61499*. The Institution of Electrical Engineers, 2001.
- Oscar Ljungkrantz and Knut Åkesson. A study of industrial logic control programming using library components. In *Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering*, pages 117–122, Scottsdale, AZ, USA, 2007.
- Oscar Ljungkrantz, Knut Åkesson, and Martin Fabian. Formal specification and verification of components for industrial logic control programming. In *Proceedings of the 4th IEEE Conference on Automation Science and Engineering*, pages 935–940, Washington DC, USA, 2008.
- M.R. Lucas and D.M. Tilbury. A study of current logic design practices in the automotive manufacturing industry. *Int. J. Human-Computer Studies*, 59(5):725–753, 2003.
- Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
- Kenneth L. McMillan. *The SMV language*. Cadence Berkeley Labs, 1999. URL <http://www.kenmcmil.com/language.ps>.
- Mostafa G. Mehrabi, A. Galip Ulsoy, and Y. Koren. Reconfigurable manufacturing systems: Key to future manufacturing. *J. Intelligent Manufacturing*, 11(4):403–419, 2000.
- David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
- Johan Richardsson and Martin Fabian. Modeling the control of a flexible manufacturing cell for automatic verification and control program generation. *J. of Flexible Service and Manufacturing*, 18(3):191–208, 2006.

Control and Plant Modeling for Manufacturing Systems using Basic Statecharts

Raimundo Moura¹ and Luiz Affonso Guedes²

¹Federal University of Piauí – UFPI

²Federal University of Rio Grande do Norte – UFRN
Brazil

1. Introduction

Based on the IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990, 1990), “**a system can be regarded as a collection of components organized to accomplish a specific function or set of functions**”. The key point in this definition is the interaction among system components. Cassandras & Lafortune (2008) discuss systems classification, especially for Discrete Event Systems (DES). In their definition, DES are systems that have discrete state space and an event-driven dynamic, i.e., the state can only change as a result of instantaneous events occurring asynchronously over time. In this context, state-based methods such as Finite State Machines (FSM) and Petri Nets have been traditionally used to describe these systems.

The automation area uses concepts of the theory of systems to control machines and industrial processes. Considering an industrial automation process based on *Programmable Logic Controllers (PLC)*, the **sensors** are installed in the plant and generate events that represent input variables to the PLC. The **actuators** are associated with the actions produced by the PLC program and represent output variables. Industrial controller programming is currently performed by qualified technicians using one of the five languages defined by IEC-61131-3 (1993) standard and who seldom have knowledge of modern software technologies. Furthermore, controllers are often reprogrammed during plant operation life-cycle to adapt them to new requirements. As a result, “*for practically no implemented controller does a formal description exist*” (Bani Younis & Frey, 2006). In general, PLC are still programmed by conventional “trial-and-error” methods and there is no written documentation on these systems.

On the other hand, *software* reusability and composability have been discussed since the 80’s, with the use of object-oriented methods (Boehm, 2006). In the Industrial area, the IEC-61499 (2005) standard allows reuse of application parts (function block, sub-application) in different applications. Software reuse is a complicated problem and depends not only on the means provided by the modeling language, but also on the overall application structure.

In the Computer Science area, several models guide the software development process such as the **Waterfall Model** (Royce, 1970), a sequential software development model in which development is seen as sequence of phases; the **Spiral model** (Boehm, 1988), an iterative software development model which combines elements of software design and prototype stages; and **agile methods**, which emerged in the 1990. Examples of the latter are: *Adaptive*

Software Development, Crystal, Dynamic Systems Development, eXtreme Programming (XP), Feature Driven Development, and Scrum. B. Boehm (2006) presents an overview of the best software engineer practices used since 1950 (decade to decade) and he identifies the historical aspects of each tendency.

In short, an application life-cycle can be divided in three phases: **Modeling - Validation - Implementation** (see Figure 1). Modeling is phase that demands more time in application lifecycle. The “Modifications” arc represents multiple iterations that can occur in software modeling processes. The “Re-engineering” arc represents the research area, which investigates the generation of a model from legacy code. Our focus is in forward engineering, which investigate the model generation from requirements specified by users.

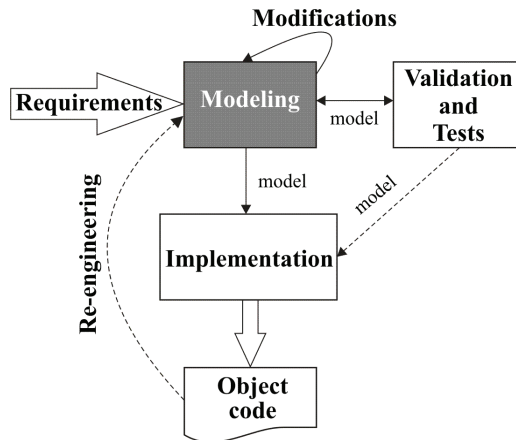


Fig. 1. Application life-cycle: overview.

In literature, there are several approaches that present methodologies, languages, and patterns for modeling industrial applications, especially for *Discrete Event Systems (DES)* (Cassandras & Lafortune, 2008). The two most common approaches are *Finite State Machines (FSM)* and *Petri nets*; both allow for formal verification of the correctness of a control system. However, despite significant research advances in recent years, these formal techniques have not been widely employed in industry (Endsley et al., 2006). We believe that such approaches are still low-level formalisms, resulting in large and unwieldy systems. The *Statecharts* formalism, described by David Harel (1987), makes the specification and design of complex DES easier. It extends conventional finite state machine with notions of **hierarchy, concurrency, and communication**.

Owing to the aforementioned problems, this work discusses a methodology for plant and control modeling and validating of the manufacturing systems that include sequential, parallel and timed operations, using a formalism based on *Statecharts*, denominated *Basic Statechart (BSC)*. For the validation phase, simulations were executed through the execution environment developed by the *Jakarta Commons SCXML Project (SCXML, 2006)*, and, as the control software model does not represent the controller itself, a translation from this model into a programming language accepted by the PLC was also carried out. In this study, *Ladder diagrams* were used because it is one of the languages defined by international IEC-61131-3 standard most widely used in industry. However, these models can be translated into any IEC-61131-3 standard language.

The remainder this work is organized as follows: Section 2 discusses about the main aspects of the *Statecharts* in modeling of automation systems and we introduce the semantic of the BSC using only characteristics relevant to the industrial area. Section 3 describes in general the methodology proposed by this contribution. In Section 4, we discuss an algorithm for translating the control model described in *Basic Statecharts* into *Ladder diagrams*, thereby enabling tests with actual PLCs. In Section 5, one typical example of application in the manufacturing area is discussed as case study to illustrate our ideas. In the last section, we conclude with a discussion about future projects.

2. Basic statecharts

Automata-based methods have been widely used to model DES, especially by the *Supervisory Control Theory* (Ramadge & Wonham, 1989). Automata represent mathematical abstractions that explicitly enumerate all the states of the system. To construct complex systems, the Automata are formally composed through systematic operations such as product and parallel composition. Moreover, they facilitate the analysis of system properties related to the validation and verification processes. However, the main drawback of the approach is inherent in the graphic representation of the model, due to the exponential growth of the number of states in the composition operations (Cassandras & Lafortune, 2008).

Statecharts formalism was described by David Harel in the 1980s and it extends conventional automata with notions of hierarchy, concurrency, and broadcast communication. Thus, *Statecharts* facilitate the specification and design of complex DES. Hierarchy and concurrency are represented through **OR-decomposition** and **AND-decomposition**, respectively. It is worth mentioning that *Statecharts* do not explicitly enumerate all the system states. Therefore, an implicit combination of the parallel states must be performed to obtain the real configuration of the model; that is, the real state of the system. Moreover, *Statecharts* have a compact graphic representation that can be translated into automata, according to the description in (Drusinsky & Harel, 1989).

The absence of a formal semantic of the original *Statecharts* makes the verification of these models very complex to carry out. In an attempt to minimize this problem, several *Statechart* variants were defined. Michael von der Beeck (1994) makes a comparison between 20 variants, and discusses a number of problems related to the original *Statecharts*. In addition, the broadcast communication of the *Statecharts* allows a triggered event in one state to affect another state that has no dependent relation with the former. Another drawback of the original *Statecharts* is that they allow interlevel transitions without imposing any constraints, a situation that can generate unstructured models.

To incorporate the advantages of the original *Statecharts* and to avoid the aforementioned problems, we propose a formalism to model DES based on *UML/Statechart* diagrams, but with a more limited syntax and semantic, denominated **Basic Statechart (BSC)**.

The *Basic Statecharts* use the syntax of *UML/Statecharts* with some variations; for example: i) absence of history connectors; ii) inclusion of input/output data channels to allow explicit communication between the components and to avoid broadcast messages in the system; and iii) the transitions are represented by the expression “[condition]/action”, where the conditions are composed using variables, data channels and the logical operators AND, OR and NOT; and, the actions allow one to change the value of these variables. The semantic of *Basic Statecharts* is more restrictive than that of *UML/Statecharts* to avoid conflict and

inconsistency in model evolution. We believe that this semantic is more appropriate for modeling industrial systems.

A **BSC** is composed of a collection of components and a **BSC component** is a structure used to model the behavior of a system element. A component can contain states, input/output channels, internal variables, and other components, which can be called subcomponents. A **data channel** is a resource used to communicate between system components. The **input data channels** are implicitly associated with internal variables and thus their values are maintained during the entire execution cycle. They can be used to change the value of guard condition from the component or external entity, such as control software or a simulation environment. The **output data channels** are also associated with internal variables; however, their values are updated only at the end of the execution cycle. They are used to publish the status of internal elements from one component to another.

The conceptual model describing the relationship between the elements that make up a BSC diagram is shown in Figure 2.

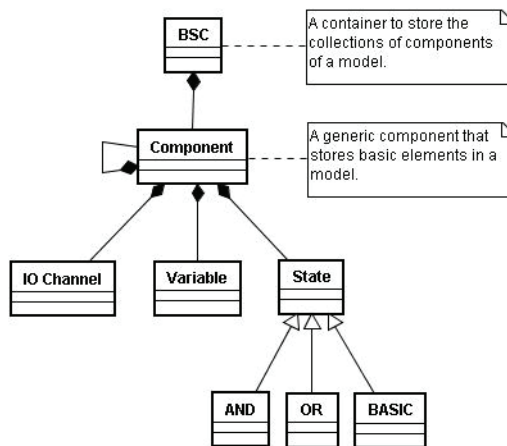


Fig. 2. Basic Statecharts: conceptual model.

The evolution of the BSC dynamic behavior is performed by sequential steps, called the *execution cycle* or *macrostep*. One constraint that is ensured by the BSC is that **a component composed of basic states can only trigger one transition in each execution cycle (macrostep)**. As with original *Statecharts*, each macrostep in BSC can be divided into several microsteps; however, the actions performed when one transition is triggered only update the variables defined in the component data area. Moreover, the BSC run accordance with definition order of the components. Thus, in an execution cycle only one component can affect the components subsequently defined in the model. This point represents a difference between the proposed approach and the Harel diagrams specified by UML. *Basic Statecharts* make the definition of validation techniques more practical, because their syntax and semantic are more constrained than those of the original *Statecharts*.

A macrostep of a BSC execution is finished when all the components have been analyzed. The BSC communication mechanism follows a *publish/subscribe pattern*: the variables associated to output channels are published in a global area, and the variables associated to input channels are consumers of these data. It is important to note that a component can be

both publisher and subscriber of a same data item. However, the published value in one step is only consumed in the next step. It is also valid for different components. Moreover, one published value can be consumed by several components in a same step, but the value of all components is guaranteed to be the same.

3. Plant and control: modeling and validation

In industrial applications, normally the controller software is verified in conjunction with a model of the plant in which it operates. So, it is necessary to obtain an accurate model to maintain fidelity with the real plant (relation one-to-one).

3.1 Plant: modeling

For plant modeling, our methodology is based on the *hybrid approach - bottom-up and top-down*. More specifically, it proposes to model the basic elements, grouping them into larger structures. This process is repeated until it generates the correct model of application. The methodology consists of three phases described as follows:

1. Modeling the basic application elements or using models already defined in a component repository;
2. Decomposing the basic states in substates, if necessary;
3. Representing all automation plant components as parallel states;

Phases 1 and 2 consist of modeling and refinements of the basic elements which compose the application. They can be run several times as an iterative process. In each iteration, we work with components which are more and more complex. Further, these components can be grouped in a repository. The third phase determines that all application components must be executed at the same time, in a parallel way, where the communication between them is made by input/output channels.

We will present how our methodology works below.

3.1.1 Basic components: patterns

For automation systems, many components follow an *On/Off pattern*, for example, valves and sensors. Figure 3-a shows the dynamic behavior of this pattern, which can be in states: "Off" or "On", and two transitions to change from state: "[g1]" from "Off" to "On" and "[g2]" from state "On" to "Off". Other components require adjustment in modeling to include new characteristics. For example: a temporary state (Wait) between the states "On" and "Off" (see Figure 3-b).

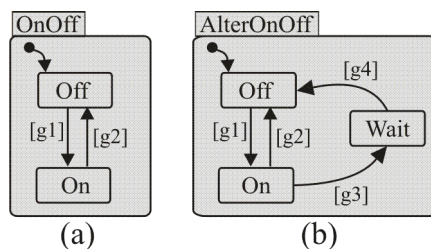


Fig. 3. On/Off patterns: basic model.

3.1.2 Cylinder component

In the manufacturing field, one of the most common components is the pneumatic cylinder that can be composed of more simple components (valves, arms and sensors) and can have displacement sensors/end-position initiators.

Figure 4 depicts a single-action cylinder with advancing controlled by the valve, return carried through springs, and one end-position sensor which is triggered when the cylinder arm gets the full advance. The generic notation “[g]/A” in a transition means that: when a guard condition g is true, the action A will be executed. Therefore, if an action in a component X1 updating one variable used in guard condition of a component X2, then we will say that: X2 depends on component X1. According to figure, the transition “[ch]/v1=1” and “[v1]/tm1=1” indicate that: the cylinder arm depends on the valve, i.e., the arm advances while the valve remains open. When the valve is closed through the action “ch=0”, the cylinder arm gets “Returned”, in function of transitions “[¬v1]/tm1=0” or “[¬v1]/v2=0”.

The cylinder arm has the following behavior: when the variable v1 gets true, the arm gets to “Advancing” in a specified time, which depends on technical characteristics and it is represented by “*” in the figure. If the valve is closed before this specified time (event tm1.tm), the cylinder arm gets to “Returned” and nothing happens to the sensor. If the event tm1.tm occurs, then the arm gets to “Advanced” and the active state of the sensor passes from “False” to “True”, implicitly. So, when the valve is closed, the arm gets “Returned” and the sensor passes from “True” to “False”.

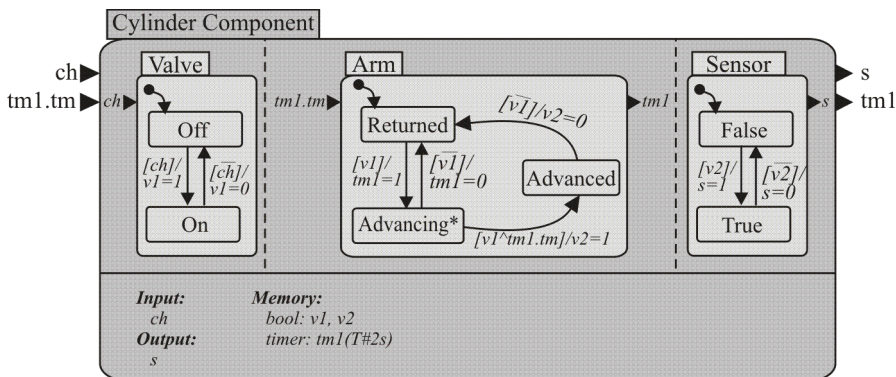


Fig. 4. Single-action cylinder: basic model.

The scenario that describes the desired operation of the cylinder is very simple: one external event allows the opening of the valve when the channel gets equal 1 ($ch=1$); then the transition “[ch]/v1=1” is run; and after the sensor detects the total advance of the cylinder-arm, the valve must be closed (data channel equals 0, i.e., $ch=0$); then the transition “[¬ch]/v1=0” is run. The events to open/close the valve represent the control police that is run by the model and define the dynamic cylinder.

3.2 Control software: modeling

In the manufacturing area, actuator components are controlled through events that are triggered by devices, such as buttons, sensors, and timers, which are defined in the control model using temporary variables. The controller is modeled through the composition of components; i.e., complex models are constructed from simpler models. The basic

components are: a) **actuators** that are modeled using components with two states: OFF and ON; b) **timers** that are modeled using components with three states: OFF, START and ON - the state "START" starts the timer and the transition "[tm1.tm]" from state "START" to state "ON" triggers the end of the timer event; and c) **variables** that are associated with sensors and temporary elements. Figure 5 shows the basic model for these elements. In this figure, g1, g2, and g3 are guard conditions. The data model area in Figure 5-c defines two Boolean variables (s1 and s2), both with the "false" value, using the syntax of the SCXML specification that was implemented by the *Jakarta Project Commons SCXML* (SCXML, 2006). This project provides a generic event-driven state machine based on the execution environment, borrowing the semantics defined by SCXML, which represents the *Statechart* diagrams by a XML file.

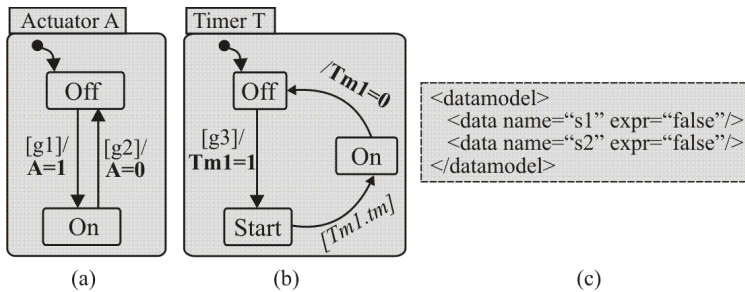


Fig. 5. Actuators: basic model.

Operational requirements of the actuators are inserted into the model as transitions between the states, in the following general form: "[guard condition] / action". The guard conditions are Boolean expressions composed of data channel and internal variables, interconnected through logical connectors \neg (negation), \parallel (disjunction) and $\&$ (conjunction). The actions can be, for example, an assignment statement to set a value in the variable and/or data channel. Therefore, operational requirements are constraints in the model to implement dependencies and/or interactions between the components. Such constraints allow us to define sequential and parallel behavior in the model; this will be described in the next subsections.

3.2.1 Sequential operation

Consider a plant composed of two actuators (A_i and A_j) that run sequentially one after the other, i.e., $A_i; A_j$. This sequence is run continuously in a cyclical way until user intervention. The sequential behavior of A_i and A_j is obtained through the execution of actions in actuator A_i , which generates internal event triggers in actuator A_j . In general, an action in an actuator can cause state changes in other actuators.

Figure 6 shows the *Basic Statechart* diagram for modeling the sequential behavior between actuators A_i and A_j discussed above. In this figure, ch_1 , ch_2 and ch_3 are input data channels; ch_1 , A_i and A_j are output data channels, and "ev" is an internal variable. Note that a same channel can be both input and output channel in a model. This is possible because the channels are associated implicitly with internal variables. These elements are used to generate the desired model behavior. In this case, the "ev" variable is used as an action by actuator A_i , which indicates the end of its actuation. It is perceived by actuator A_j , which starts its operation, generating the sequential behavior between them. Note that the data

model area is not represented in the figure. At the end of A_j actuation, data channel ch_1 is updated, generating the cyclical behavior of the model. In its initial configuration, all the actuators of the model are set to "Off". The system starts its operation when data channel ch_1 is equal to 1 (Boolean value "true"), a situation that can be simulated when the operator pushes a "start" button on the Interface Human-Machine (IHM), for example.

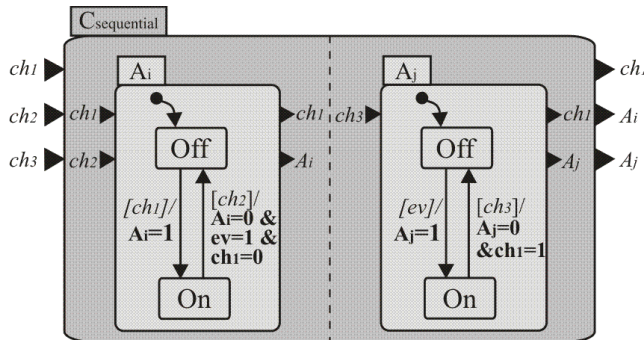


Fig. 6. Control model: sequential operation.

3.2.2 Parallel operation

Parallelism, an inherent characteristic of original *Statecharts*, is accomplished through AND-decomposition. However, the component synchronism demands additional mechanisms. Consider a plant composed of three actuators (A_i , A_j and A_k), where A_i and A_j run in parallel, but A_k can only run after the execution of the two first components, i.e., $(A_i \mid A_j); A_k$. This sequence is run continuously in a cyclical way until operator intervention. The parallel behavior of A_i and A_j is obtained naturally; however, internal variables must be used to generate internal event triggers in actuator A_k to indicate the end of execution in other actuators. Thus, A_k must wait for these updates to start its operation. After the A_k run, these internal variables must be updated to allow the execution of a new cycle in the system. Figure 7 shows the *Basic Statechart* diagram for modeling the parallel behavior between the aforementioned actuators. In this figure, ch_i ($i = 1..5$) are input data channels, A_i , A_j and A_k are output data channels, ev_i and ev_j are internal variables. These elements are used to generate the desired application behavior. In this case, the variable ev_i is updated as an action by actuator A_i , indicating the end of its actuation, and the variable ev_j is updated to indicate the end of A_j actuation. These updates are perceived by actuator A_k , which starts its operation, generating the synchronism between them. At the end of A_k actuation, the ev_i and ev_j must be "reset" to generate the cyclical behavior of the model. In its initial configuration, the model must have all actuators set to "Off".

3.2.3 Timed operation

Timers and counters are quite common in industrial applications; for example: i) an actuator must execute for a specific time; ii) an actuator must execute only after a specific time; iii) the system must execute k times before triggering an alarm; and so on. Timers and counters are modeled through basic components and their current values can be used to set the guard conditions of the transitions in BSC. Furthermore, they can be started and/or reset by some action of the model.

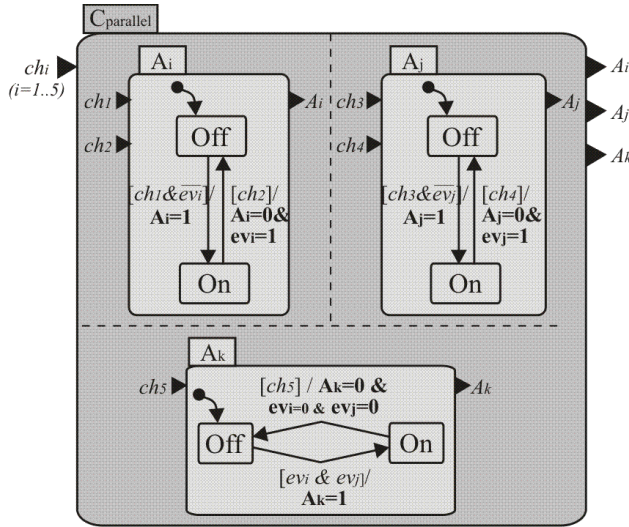


Fig. 7. Control model: parallel operation.

Timers are controlled by a global real-time clock that executes in parallel to the system model, and they are updated only at the beginning of each execution cycle. Thus, when a timer is enabled in a component, the timing process is initiated in the next execution cycle. When the timer reaches or surpasses its specified limit, an internal variable tm is made true ($tm = true$) to indicate end of timing. In the timer, creating must define the time limit value in time units.

Consider a plant composed of an actuator A_i and a timer T_k , where A_i must act for t seconds before turning off. Figure 8 shows the *Basic Statechart* for modeling the temporal behavior of actuator A_i , controlled by timer T_k . In this figure, ch_1 and ch_2 are input data channels used to start the operation of actuator A_i and of timer T_k , respectively, and $tk.tm$ is an input data channel used to indicate the timeout of T_k . It is important to mention that the timers are updated as a global action of the model, and the timer T_k is started when action $tk = 1$ is executed.

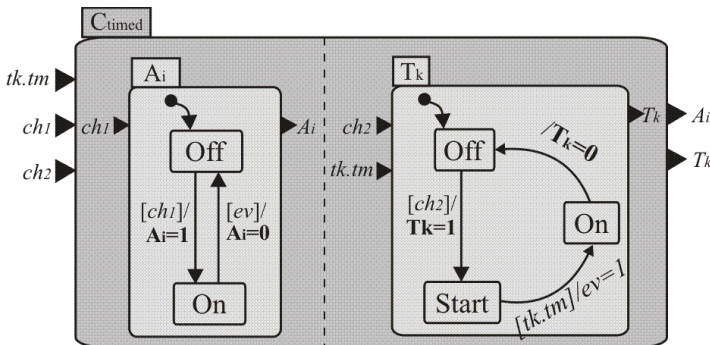


Fig. 8. Control model: timed operation.

The guard condition “ev” used to turn off actuator A_i becomes true when timer T_k reaches or surpasses the specified limit (condition $tk.tm$). Thus, the constraint that defines that actuator A_i must execute for a specific time is ensured.

3.3 Control software: validation

The approach for modeling the control software discussed in Section 3.2 maintains the description and specification aspects built into the *Basic Statechart* model. Transitions, guard conditions, and implicit actions are used to describe system constraints. Thus, the approach allows us to analyze some controller properties using the reachability tree of the formal model. Moreover, simulated environments can be used to validate the control model along with the plant model.

The reachability tree of the model allows us to analyze a number of properties, such as: i) **reinitiability** – for each cfg_i state configuration reached from the initial cfg_0 configuration, is it possible to return to cfg_0 by a sequence of events? ii) **vivacity** – does the controller act in all of the components in the model? iii) **deadlock** – is there a cfg_i state configuration in which progress cannot be made because no transition can be triggered?

Masiero et al. (1994) propose an algorithm to create a reachability tree for *Statecharts*. Here, we briefly discuss an adaptation of this algorithm to analyze the aforementioned structural properties. This algorithm was implemented using Java language and the SCXML execution environment, with the following modifications:

- The set that contains all possible transitions for a given configuration includes only the transitions with events controlled by an external agent, and with timed events triggered automatically by the components.
- To obtain a new configuration of the model by triggering a transition, the internal variables are implicitly updated and, therefore, can trigger other transitions automatically in the model. This characteristic decreases the number of states produced in the reachability tree.
- The part of the algorithm that describes the history connectors is completely excluded, because *Basic Statecharts* do not include such characteristics.

The use of this algorithm allows a formal analysis of system behavior (control + plant) to verify and validate a number of properties. It is important to note that a plant model is required, and it may be represented in a given formalism; for example, automata, Petri net or *Statecharts*. Moura et al. (2008) propose a systematic procedure for modeling complex plants using *Statecharts* and discuss some aspects of control modeling. However, they presented only a descriptive view of that process.

In this work, we chose *Basic Statecharts* to model plant behavior, without losing generality. Therefore, the system (control + plant) can be described as parallel composition between the controller and plant. The main advantage of this approach is that sensor and actuator characteristics become internal events of the system. Thus, the intrinsic properties of the system, such as **reachability**, **deadlock**, and **reinitiability** become intrinsic and extrinsic properties of the controller.

Another advantage of this approach owes to the fact that it maintains controller and plant functionality explicitly separated. Here, unlike other approaches, such as the R & W approach (Supervisory control), the controller synthesis produces more compact models. In the next section we present an algorithm for translating the control model described in *Basic Statecharts* into one PLC language (in this case, *Ladder diagram*).

4. Control software: implementation

Given that the control model does not represent the controller software itself, the translation from this model into a programming language accepted by the PLC must also be performed. *Ladder* diagrams were used because it is one of the languages defined by international IEC-61131-3 standard most widely used in industry. The translation is performed systematically by a method that analyzes one component at a time, according to its type (**actuator** or **timer**).

The states (“OFF” and “ON”) in the actuators are represented in the *Ladder* through auxiliary contacts (*flip-flop Reset* and *flip-flop Set*), respectively. Each control model transition results in a “rung” of the *Ladder*, as follows: the source state must be added to the condition, and the target state represents the action that must be executed. Let *A* be the generic actuator shown in Figure 5-a, where transitions “[*g1*]/*A*=1” and “[*g2*]/*A*=0” generate lines 3 and 4, respectively, of the *Ladder diagram*, as shown in Figure 9. In this figure, *c1*, *c2*, and *c3* are auxiliary variables that are computed from the guard conditions of the model (i.e., *g1*, *g2*, and *g3*, respectively). This mapping is made because the guard conditions can be complex.

The timers were translated as follows: one “rung” to transition from the “OFF” to “START” state, which allows us to start up the timing; one “rung” to specify the timer itself, with one element that indicates the end of the specified time, which can be used in other *Ladder* lines,

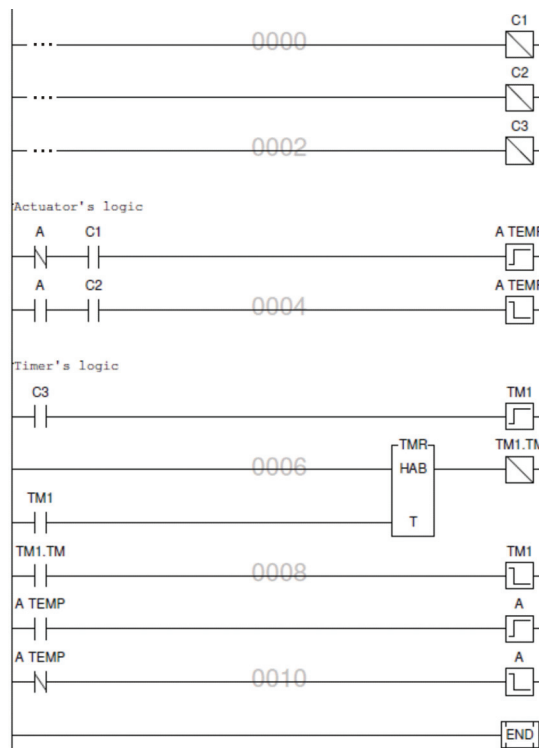


Fig. 9. Actuators: Ladder diagram.

according to the application; and another “rung” to reset the timer. The generic timer shown in Figure 5-b generates lines 5 to 8 of the *Ladder diagram* (see Figure 9). In this figure, the parameters “HAB” and “T” of the block TMR represent identifiers used to set up as follows: HAB lets it enable/disable, and T lets us define the time limit value of this block. The variables that represent the sensors and or auxiliary contacts can be freely used in the guard conditions and actions of the *Ladder code*, according to the transitions of the model. However, as the guard conditions of the transitions (in each *Ladder line*) must be guaranteed by at least one *PLC-scan cycle*, all conditions must be evaluated and stored in auxiliary variables at the beginning of each *PLC-scan cycle* (see lines 0, 1 and 2 in Figure 9).

Moreover, it is important to note that to avoid non-determinism in the system, the guard conditions for a same source state must be mutually exclusive. This constraint can be established during model building and the user can be notified by warning messages. But, as the conditions must be mutually exclusive to a same source state, these *Ladder lines* specifically cannot be generated in any order, because inconsistencies can occur in one *PLC-scan cycle*; for example, turning on/turning off an actuator. To avoid such inconsistencies, the temporary state of the actuators must be stored in auxiliary variables, and at the end of the cycle, these variables must be updated for the corresponding outputs (see lines 9 and 10 in the Figure 9).

Algorithm 1. Translation from the control model into a Ladder diagram

```

{Let there be n actuators, m timers, t transitions}
{Guard conditions analysis}
for i = 1 to t do
    Compute guard(i) {Guard condition of the i-th transition}
end for
{Actuator's logic}
for i = 1 to n do
    for j = 1 to T[Ai] do
        if target(j) = Ai.ON then
            AiTemp.set := source(j) AND guard(j)
        else
            AiTemp.reset := source(j) AND guard(j)
        end if
    end for
end for
{Timer's logic}
for i = 1 to m do
    Tmi.set := guard(enableTimer(Tmi))
    CreateTimer(Tmi, limit(Tmi)) {Function block: Timer}
    tmi.tm := Tmi.enable() AND Tmi.timeout()
    Tmi.reset := tmi.tm
end for
{Update actuators from temporary variables}
for i = 1 to n do
    Ai.set := AiTemp
    Ai.reset := ¬AiTemp
end for

```


The complete algorithm used to translate the control model into *Ladder code* is presented in Algorithm 1. In this algorithm, some terms have been used to facilitate the understanding, such as:

- **guard(t)** is the guard condition of the t-th transition;
- **source(t)** is the source state of the t-th transition;
- **target(t)** is the target state of the t-th transition;
- $T[A_i]$ is number of transitions of actuator A_i ;
- $A_i.ON$ is a constant to represent the 'ON' state of actuator A_i ;
- **enableTimer(Tmi)** is the transition that allows us to start up the timing of the i-th timer;
- **Tmi.limit(<value>)** is the time limit of the i-th timer;
- **Tmi.enable()** is a function to indicate if the i-th timer is enabled;
- **Tmi.timeout()** is a function to indicate when the i-th timer reaches the end of the specified time.

5. Case study: manufacturing cell

This section presents a case study that realizes a simulation of a manufacturing cell (see Figure 10-a), which is a typical example of the manufacturing sector where the devices can run in a simultaneous mode. This example is well explored in *Supervisory Control Theory* by Queiroz & Cury (2002). The problem with these systems is the need for synchronization points between parallel blocks.

The execution flow, with a possible operation of the devices for this system, is shown in Figure 10-b. It is interesting to note that the four device actuators can run simultaneously and that the table must be run only after the execution of these devices. Thus, a synchronization point between devices and the table must be created to enable proper system operation.

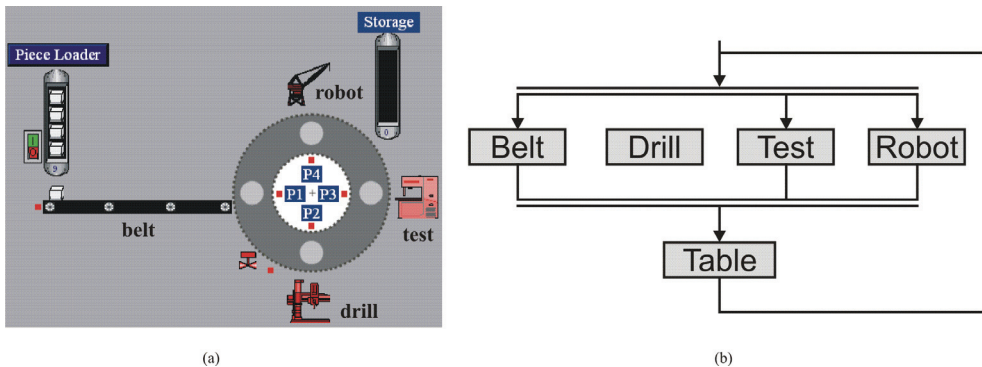


Fig. 10. Manufacturing cell: simulation environment.

Consider the run scenario described below:

- **BELT:** If there is a piece in the input buffer (initial position of the belt) and none in position P1, the belt must be turned on; later, when the piece is at position P1 the belt must be turned off. The *if ... then clauses* of this specification are:
 - **If** inputbuffer & $\neg P1$ **then** BeltOn;

- **If P1 then** BeltOff;
- **DRILL:** If there is a piece in position P2, the drill and a timer component timerT1 must be turned on; at the end of timeout, the drill must be turned off. The *if ... then clauses* of this specification are:
 - **If P2 then** DrillOn & tm1On;
 - **If tm1.tm then** DrillOff;
- **TEST:** If there is a piece in position P3, the test and a timer component timerT2 must be turned on; at the end of timeout, the test must be turned off. The *if ... then clauses* of this specification are:
 - **If P3 then** TestOn & tm2On;
 - **If tm2.tm then** TestOff;
- **ROBOT:** The robot removes a piece from position P4, and stores it. If there is a piece in position P4, the robot and a timer component timerT3 must be turned on; at the end of timeout, the robot must be turned off. The *if ... then clauses* of this specification are:
 - **If P4 then** RobotOn & tm3On;
 - **If tm3.tm then** RobotOff;
- **TABLE:** The table rotation is controlled by the single-action cylinder and the total advance of the cylinder arm generates a 90 degree turn. Thus, after the execution of the four devices, the cylinder must be activated to obtain a new system configuration. The return of the cylinder-arm should occur when the sensor detects the total advance of the cylinder-arm. The *if ... then clauses* of this specification are:
 - **If BeltEnd & DrillEnd & TestEnd & RobotEnd then** ValveOn;
 - **If SensorOn then** ValveOff;

The belt model follow the *Alter On/Off pattern* (see Figure 3-b), whereas the drill, the test, and the robot models follow the *On/Off pattern* (see Figure 3-a). The table behavior is modeled through of single-action cylinder (see Figure 4). In each table position, there is one sensor for simulating piece in the place. Thus, the complete plant model is generated by representation, in parallel way, of the four devices and the cylinder, as can be shown in Figure 11.

Other constraints imposed on the model are:

1. Each device must execute only once before a table rotation;
2. If in a configuration there is no piece in the input buffer or in positions P2, P3, and P4, then the belt, the drill, the test, and the robot must not be turned on;
3. The table rotation must only be performed if there is at least one piece in positions P1, P2, or P3.

The inclusion of these constraints in the controller model is carried out by determining new transitions between states and/or changes in the guard conditions of the existing transitions. Initially, to create the control model for this case study, extra variables must be included to ensure synchronism between the devices and, therefore, the constraint imposed on table rotation, i.e., the table cannot rotate while the devices are running. In this case, the variables E1, E2, E3, and E4 indicate the "end-of-operation" of the belt, drill, test, and robot, respectively. These variables must be set to "true" for each of the devices. According to cell operation, the table must only be rotated when all of devices have concluded their operations, i.e., when the variables $E_i = \text{true}$ ($i = 1, \dots, 4$). After the table rotates 90 degrees, these variables must be reset to allow new operations in the system. These variables are also used in the transitions to turning on/turning off the actuators; for example, the drill must only be turned on if the E2 control variable is equal to "false".

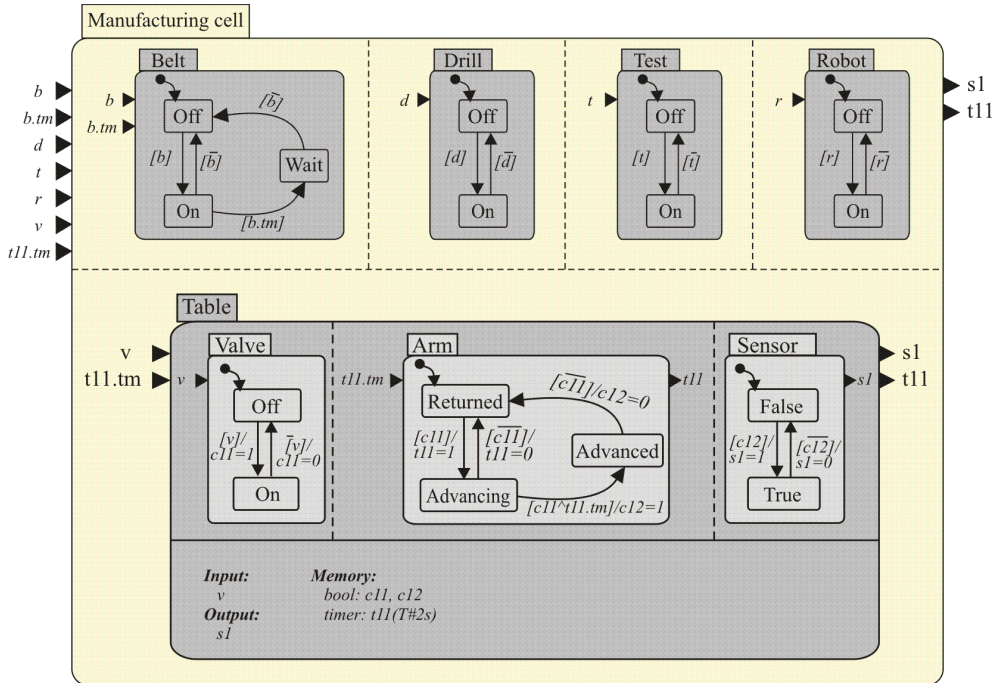


Fig. 11. Manufacturing cell: plant model.

Extra transitions to ensure constraints 2 and 3 must be included in the model. For example, if there is no piece in position P2, then the drill must not be turned on, but the E2 variable must be set to “true” to indicate end-of-operation of the phase. Similar ideas are applied to other actuator devices. In the table model, if there is no piece in positions P1, P2 or P3, then the table must not rotate (constraint 3); however, variables E1, E2, E3, and E4 must be set to “false” to allow new operations in the devices. Thus, if there is no piece in the manufacturing cell, the model will continually alternate the value of E1,...,E4 between

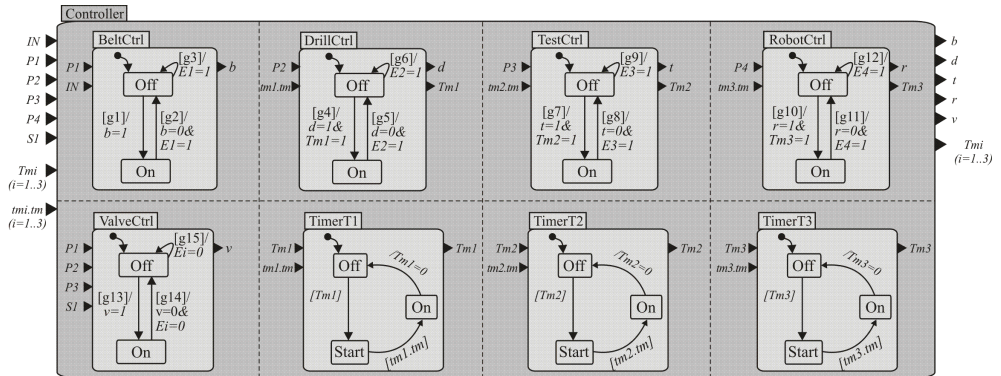


Fig. 12. Manufacturing cell: control plant.

“false” and “true”. The complete BSC model of the controller software is shown in Figure 12. Guard conditions g_1, g_2, \dots, g_{15} are presented in Table 1, where the variable IN indicates the presence or absence of a component in the input buffer. Note that the data area is not represented in the figure, but the IO channels can be easily identified; E_i ($i = 1, \dots, 4$) are internal variables, and P_1, \dots, P_4 , IN, S1 represent sensors installed in the plant.

g_1	$\neg P_1 \ \& \ \neg E_1 \ \& \ IN$
g_2	P_1
g_3	$\neg P_1 \ \& \ \neg E_1 \ \& \ \neg IN$
g_4	$P_2 \ \& \ \neg E_2$
g_5	$tm1.tm$
g_6	$\neg P_2 \ \& \ \neg E_2$
g_7	$P_3 \ \& \ \neg E_3$
g_8	$tm2.tm$
g_9	$\neg P_3 \ \& \ \neg E_3$
g_{10}	$P_4 \ \& \ \neg E_4$
g_{11}	$tm3.tm$
g_{12}	$\neg P_4 \ \& \ \neg E_4$
g_{13}	$E_1 \ \& \ E_2 \ \& \ E_3 \ \& \ E_4 \ \& \ (P_1 \ \ P_2 \ \ P_3)$
g_{14}	$S1$
g_{15}	$E_1 \ \& \ E_2 \ \& \ E_3 \ \& \ E_4 \ \& \ \neg P_1 \ \& \ \neg P_2 \ \& \ \neg P_3$

Table 1. Controller: guard conditions.

This example is composed of the belt with three possible states, three devices with two states each, and one cylinder linked to the table, which also has three states. The model with no control has 72 states, i.e., $3 \times 2 \times 2 \times 2 \times 3 = 72$ distinct configurations, and the controlled model (control + plant) has 210 different states, in function of three timers included in the control model for simulating the processes of drilling, testing and moving the piece to storage. However, these 210 configurations act only in 26 distinct configurations, where: i) 24 possibilities of actuation of devices: $3 \times 2 \times 2 \times 2 = 24$ with the table in position stop (cylinder in configuration [*Off, Returned, False*]); and ii) 2 possibilities for rotating the table with the four devices in state Off. The reachability tree analysis has shown that the model ensures the properties of *reinitiability*, *vivacity*, and that there is no *deadlock*. But, this analysis is out of scope of this work.

6. Conclusion

In this work we presented a methodology for systematizing the process of plant and control modeling of manufacturing systems. Our proposal uses a formalism based on *Statecharts* diagrams, called **Basic Statecharts (BSC)**. The plant modeling has three phases which can be executed as many times as necessary. In general, this methodology represents a hybrid approach - *bottom-up and top-down*, allowing components reuse and keeping a one-to-one relation between plant and model (i.e., it is faithful to the actual system). The control model is generated also using **Basic Statecharts**. Thus, the main contributions of this work are the following:

- A methodology to model plants and industrial control logics using **Basic Statecharts**;

- A procedure to integrate plant and control models in order to analyze and/or validate several structural proprieties of the modeled system, such as **deadlock absence**, **vivacity**, and **reinitiability**. This is very important in the project phase of every industrial controller;
- An algorithm to translate the control logics described in **Basic Statecharts** into *Ladder diagrams*.

One typical example of the manufacturing application was described as a case study to illustrate our proposal.

A prototype using Java language is currently being developed to create and simulate models generated by our methodology. The aim is to test how much easier and natural the creation of industrial applications will become, as well as to produce more “user-friendly” documentation for the designers, giving more autonomy to the development and maintenance teams.

7. References

- Bani Younis, M. & Frey, G. (2006). UML-Based Approach for the Reengineering of PLC Programs, in *Proceedings of 32nd Annual Conference of the IEEE Industrial Electronics Society (IECON'06)*, pp. 3691–3696, Paris, France, November, 2006.
- Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement, *Computer*, Vol. 21, Issue 5 (May 1988), pp. 61 – 72, ISSN:0018-9162
- Boehm, B. (2006). A View of 20th and 21st Century Software Engineering, in *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, pp. 12–29, New York, NY, USA: ACM Press, 2006.
- Cassandras, C. & Lafortune, S. (2008). *Introduction to Discrete Event Systems - Second Edition*, Springer Science, ISBN-13: 978-0-387-33332-8, New York, (USA).
- Drusinsky, D. & Harel, D. (1989). Using Statecharts for Hardware Description and Synthesis, *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 7, pp. 798–807.
- Endsley, E.; Almeida, E. & Tilbury, D. (2006). Modular Finite State Machines: Development and Application to Reconfigurable Manufacturing Cell Controller Generation, *Control Engineering Practice*, Vol. 14, No. 10 (October 2006), pp. 1127–1142.
- Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8, No. 3 (June 1987), pp. 231–274.
- IEC-61131-3 (1993). International Electrotechnical Commission. Programmable Controllers Part 3, Programming Languages.
- IEC-61499-1 (2005). International Electrotechnical Commission. Functions Blocks Part 1, Architecture, Geneva: IEC, 2005.
- IEEE Std 610.12-1990 (1990). Standard Glossary of Software Engineering Terminology, <http://ieeexplore.ieee.org/ISOL/standardstoc.jsp?punumber=2238>.
- Masiero, P.; Maldonado, J. & Boaventura, I. (1994). A reachability Tree for Statecharts and Analysis of Some Properties, *Information and Software Technology*, Vol. 36, No. 10, pp. 615–624.
- Moura, R.; Couto, F. & Guedes, L. (2008). Control and Plant Modeling for Manufacturing Systems using Statecharts, in *IEEE International Symposium on Industrial Electronics (ISIE 2008)*, Cambridge, UK, July 2008, pp. 1831–1836.

- Queiroz, M. & Cury, J. (2002). Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell, *In Proceedings of the 6th International Workshop on Discrete Event Systems*, IEEE Computer Society, pp. 377-382.
- Ramadge, P. & Wonham, W. (1989). The Control of Discrete Event Systems, *in Proceedings of the IEEE*, Vol. 77(January, 1989), pp. 81-98.
- Royce, W. W. (1970). Managing the Development of Large Software Systems, *in Proceedings of IEEE WESCON*, pp. 1-9.
- SCXML (2006). The Jakarta Project Commons SCXML, <http://jakarta.apache.org/commons/scxml/>.
- Von der Beeck, M. (1994). A Comparison of Statecharts Variants, *in ProCoS: Proceedings of the Third International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems*. London, UK: Springer-Verlag, pp. 128-148.

The Java based Programmable Logic Controller. New Techniques in Control and Supervision of a Flexible Manufacturing Cell.

Ramón Piedrafita and José Luis Villarroel

*Department of Computer Science and Systems Engineering, University of Zaragoza
Spain*

1. Introduction

In this chapter, we explain new techniques and technologies applied to the control of a flexible manufacturing cell. We have proposed a new control platform: a Java based Programmable Logic Controller (PLC). The Java PLC comprises several modules where the real time control, the communication with industrial fieldbuses and the supervision via web technologies have been developed. This control architecture (Piedrafita & Villarroel 2006) and development environment is based on Petri Nets (PNs), Sequential Function Charts (SFCs) and the Real Time Java programming language. Our objective is to explore the application of new control techniques of manufacturing systems, and to test the use of the Java programming language as a platform for those techniques. To demonstrate the practical utility of the techniques, we have applied them to the control of a flexible manufacturing cell.

This research follows earlier studies at the University of Zaragoza on the software implementation of PN. In those studies, Ada95 was the implementation language (García & Villarroel 1996). For the current study, Java was chosen for the following reasons:

- The possibility of executing the same code on different platforms.
- To compare the Ada95 and Java implementations using the same concurrent and real-time characteristics.
- Java is a language that is beginning to be used in the development of control and embedded systems.
- Java has a real time extension that allows the necessary time predictability required in these types of applications.

From the perspective of the software implementation of Discrete Event Control Systems, we have translated into the Java language some classic PN implementation techniques such as Enabled Transitions or Representing Places. We have also developed SFC implementation techniques such as the Deferred Transit Evolution Model and Immediate Transit Evolution Model.

In the execution of a Petri net, a task, the coordinator, makes the firing of transitions and updates the marking. We propose a new PN implementation technique called concurrent coordinators, in which centralized and decentralized ideas are merged. The net is decomposed into several subnets following, for example, a functional criterion, and a

different coordinator implements each subnet in a centralized way. The communication between the different subnets is made using communication places that are implemented by monitors that use synchronized methods for marking and unmarking.

The Java PLC can load PNs in PNML (Billington, Christensen et al. 2003) and can load SFCs in the PLCopen XML format (Open 2005). In the input/output module we have developed libraries for communication with bus Interbus, CANopen and Modbus TCP/IP.

The hardware of the Java PLC is based on a PC equipped with several fieldbus master cards, where the input/output modules of the machines are connected. The software of the control system is based on a real-time control program that has several threads responsible for the concurrent execution of PN or SFC programs. The threads communicate through monitors.

The development of this open platform has involved the development of an open framework of Java classes. This includes classes for the PN and SFC implementation, from the basic classes that model *places*, *transitions*, and the structure of a Petri net to the classes that execute the PN, and classes that allow communication with several fieldbuses. There are also classes for remote supervision via the RMI protocol or web Supervision.

A practical application of the proposed approach, the control of a flexible manufacturing cell, was developed at the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain.

The remainder of this chapter is organized as follows. In Section 2 we present the characteristics of the Java language relevant to the implementation of control systems and we present the Java PLC structure. In section 3 we review the different techniques for the implementation of PNs and SFCs. The flexible manufacturing cell is described in Section 4. In Section 5 the communication with the controlled system is explained. Section 6 describes the structure of the proposed control architecture and in section 7 we show a new technique: the Execution Time Controller. In Section 8, some details of the software implementation in the Java language are presented. The development environment we have created is described in Section 9 and, in Section 10 we present conclusions and suggest future lines of research.

2. Java based programmable logic controller

This research follows earlier studies at the University of Zaragoza on the software implementation of PN. In those studies, Ada95 was the implementation language (García & Villarroel 1996).

Java has some of the characteristics required for the implementation of control applications, including the concurrent execution of threads; however, Java also has characteristics that impede its use as a programming language for control applications:

- Although there exists compilers, Just in Time compilers, Virtual Machines with on-the-fly recompilation, Java mainly is used like interpreted language.
- Programs written in Java link classes dynamically; thus, the load of remote classes can delay the execution of the program.
- Java uses dynamic memory to create objects as needed, and the time to create an object depends on the memory state.
- A high-priority thread performs the garbage collection, the process of automatically freeing objects that are no longer referenced by the program.

- The Java scheduler works with fixed priorities, but there is no guarantee that the highest-priority thread is running. To avoid starvation, the thread scheduler might choose to run a lower-priority thread. The thread priority affects the scheduling policy for efficiency purposes only.

The first characteristic affects the run-time performance only, but the others imply time unpredictability, which prevents Java from being used in the development of real-time systems. These problems have been solved in the *Real Time Java Specification* (RTJS) (Bollella and Gosling 2000.) which has been implemented in some platforms, such as JamaicaVM (Aicas 2007) which includes the following:

- The Real Time Virtual Machine executes "bytecodes," but programs are also compiled to machine code, which increases the speed of execution.
- New thread classes, `RealtimeThread` and `NoHeapRealtimeThread`, which are unaffected or at least less heavily affected by garbage collection activity.
- New memory classes (`InmortalMemory` and `ScopedMemory`) that are not under the control of the garbage collector.
- A true fixed-priority pre-emptive scheduler that has an expanded range of priorities.
- The incorporation of classes for the treatment of asynchronous events and to manage the asynchronous transfer of control.
- The ability to work with high-resolution real-time clocks.

These characteristics of real time Java provide the basic tools for the development of control applications. This has enabled the development of a platform for the real time implementation of discrete event control systems based on the Java language. We call this platform the Java PLC. Petri nets are used as a tool for modelling and implementing discrete event systems, and also their programmed implementation: the Sequential Function Charts. The Java PLC should be able to execute PNs or SFCs in real time. To develop the control function, it should be able to communicate with fieldbuses in such a way that it can read the signals of the sensors and write signals to actuators. It should also be able to communicate with the plant operators providing information about the controlled system.

With these objectives, and in order to carry out the development correctly, the Java PLC has been structured in several modules, each one responsible for a function (control, communication, supervision...) and able to exchange information one with another.

The Java PLC comprises several modules (see Fig. 1):

- The control module is in charge of executing the PLC program.
- The input/output module is in charge of communicating with several fieldbuses like Interbus, CANopen or Modbus TCP/IP.
- RMI communication module. This allows communication with other Java virtual machines and for the Web Server to exchange information with the control module.
- Web Server. This module is responsible for providing information about the state of the program and the input / output variables, allowing remote visualisation of the state of the program.

3. The control module

The control module is responsible for executing the PLC programs. These programs can be run using Petri nets in the PNML format or in the Sequential Function Chart language in the PLCopen XML format. The execution in the PLC is carried out in real time as a high-priority task.

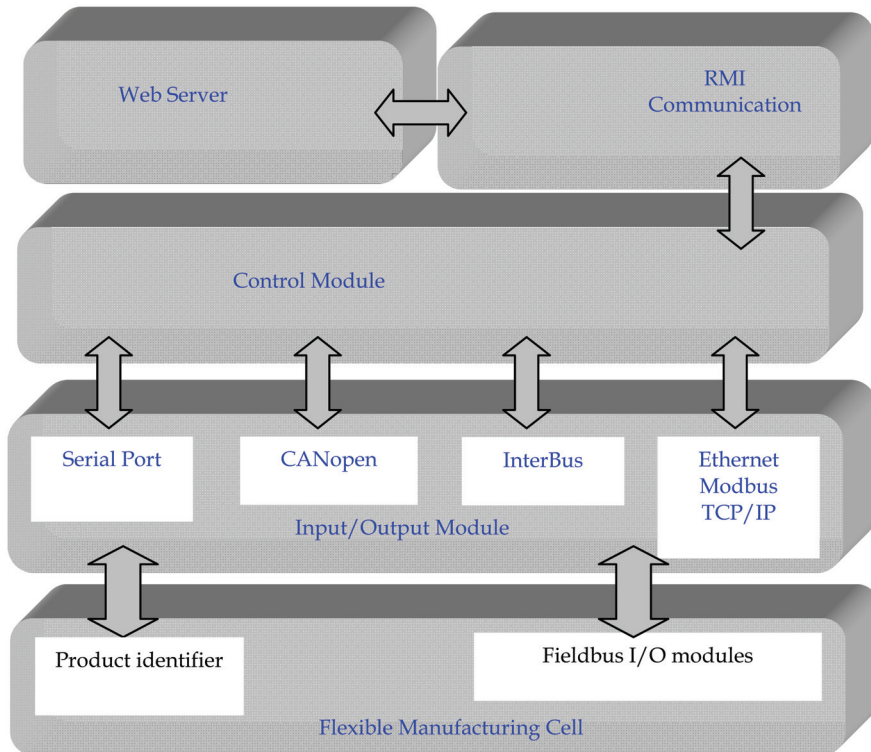


Fig. 1. Java based Programmable Logic Controller.

Currently most industrial PLCs run their programs in an interpreted and centralized manner. The PLC reads the inputs, executes the program (i.e. runs an interpreter of SFCs, also called coordinator in this paper) and writes the outputs.

In the execution of the program it is necessary to determine which transitions can fire, and then fire them so that the state of the SFC (or PN) will evolve. This will also include the actions programmed in the steps. The algorithm to determine which transitions are enabled and can fire is important because it introduces some overhead in the controller execution and the reaction time is affected. In the Java PLC we have implemented several algorithms in which different enabled transition search techniques are developed:

- Brute Force (BF). PN implementation technique.
- Deferred transit evolution model (DTEVM). SFC implementation technique.
- Immediate transit evolution model (ITEVM). SFC implementation technique.
- Static Representing Places (SRP). PN implementation technique.
- Enabled Transitions (ET). PN implementation technique.

In the Brute force algorithm all the transitions are tested for firing. Brute Force algorithms do not try to improve the search of enabled transitions. Works such as (Peng & Zhou 2004) (Uzam & Jones 1996) (Klein, Frey et al. 2003) belong to this implementation class.

The IEC-61131 standard is not very precise in the definition of the SFC execution rules. Different execution models have been proposed to interpret the standard. As with BF, in the

Immediate Transit Evolution Model (ITEVM) algorithm all the transitions are tested for firing (Hellgren, Fabian et al. 2005). However, the Deferred Transit Evolution Model (DTEVM) (Hellgren, Fabian et al. 2005) only performs the testing of the transitions descending from the marked places, improving in this way the Brute Force operation.

In (Lewis 1998) the IEC-61131 standard is interpreted and the following tasks are proposed to run an SFC:

- Determine the set of active steps
- Evaluate all transitions associated with the active steps
- Execute actions with falling edge action flag one last time
- Execute active actions
- Deactivate active steps that precede transition conditions that are true and activate the corresponding succeeding steps
- Update the activity conditions of the actions
- Return to step 1

These tasks are implemented in the DTEVM algorithm. In DTEVM, the transition conditions of all transitions leading from marked places are evaluated first. Then, the transitions found to be fireable are executed one by one. In ITEVM, the transition conditions of all transitions are evaluated one by one. In the case of a transition condition being true, i.e., the corresponding transition being fireable, this transition is fired immediately.

In the Static Representing Places (SRP) algorithm, only the output transitions of some representative marked places are tested (Colom, Silva et al. 1986). Each transition is represented by one of its input places, the Representing Place. The remaining input places are called synchronization places. Only transitions whose Representing places are marked are considered as candidates for firing.

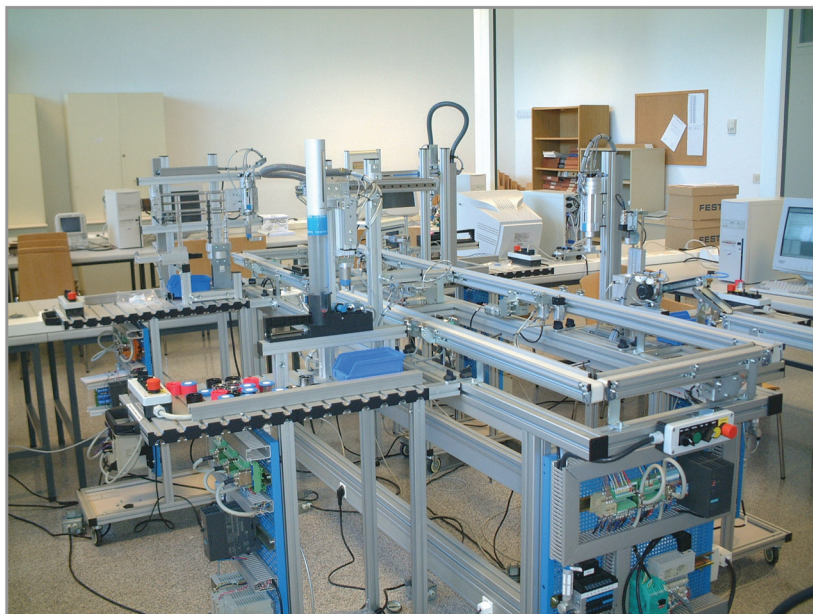


Fig. 2. Flexible manufacturing Cell.

In the Enabled Transitions algorithm, only totally enabled transitions are tested. A characterization of the enabling of transitions, other than marking, is supplied, and only fully enabled transitions are considered. This kind of technique is studied in works such as (Silva & Velilla 1982) (Briz. 1995).

In the implementations developed in the present study, the program loads the net structure from a XML file that is generated by a Petri net editor; thus, the implementation is independent of the net and is an *interpreted* implementation. In the execution of a Petri net, a thread called the “coordinator” is responsible for the firing transitions and updates the state of the net (marking), this being a *centralized* approach. Furthermore, we have introduced centralized techniques into decentralized implementations, thereby creating a new technique called *concurrent coordinators*. The application can run several coordinators simultaneously by executing a sub-net for each subsystem.

4. The flexible manufacturing cell.

The practical application of the ideas presented in this paper were tested using a flexible manufacturing cell installed at the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain, for research and teaching purposes.

The manufacturing cell carries out a complete production process making various types of pneumatic cylinders. The orders are composed by a rectangular base (white or black) and three cylinders produced in the first production ring, being able previously to select both the base and the cylinders. It also has two storage areas, one intermediate store for the manufactured pneumatic cylinders and the other for orders. The term “flexible” means that the cell can manufacture any component type at any moment.

There are two types of base, white and black, and six types of pneumatic cylinders divided into two groups, cylinders with and without a cap.

The cylinders with a cap are made up of a sleeve with a closed cap. These pieces are already manufactured and require no handling apart from identification and storage. The components without a cap are cylinders simple effect pneumatic cylinders made up of a piston, a recoil spring and a head.

There are three types in both groups: black, red and metallic. The cylinders without cap have two sorts of piston, metallic and plastic. The metallic piston is narrower and shorter than the plastic, and is only included in the black pieces, while the plastic piston is mounted in the red or metallic cylinders. The composition of the cylinders without cap is shown in Fig. 3.

The manufacturing cell is composed of a set of stations for the production and storage of pneumatic cylinders and is divided into two zones (Fig. 4): (a) the production zone (stations 1, 2, 3, and 4, and the transport 1), and (b) the product expedition zone (station 6 and storage station 7, two robots, and the transport 2). Station 5 is an intermediate storage area between the two zones.

Transport 1 (left hand side of Fig. 4) is responsible for moving the pallets in the production zone between the stations and is above the pallets where the operations are carried out. This transport connects stations 1, 2, 3 and 4 which carry out the following operations:

- Station 1: identifies and positions the sleeves for cylinders without cap or those with cap and previously manufactured.
- Station 2: assembles the corresponding piston with the sleeve type and next assembles the coil spring.

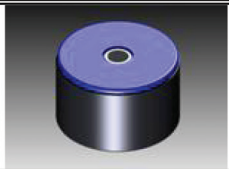
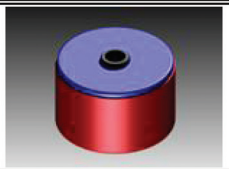
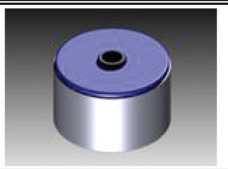



Pneumatic cylinders			
Sleeve	Black	Red	Metallic
Piston			
Coil spring	Standard	Standard	Standard
Cap	Standard	Standard	Standard

Fig. 3. Types of Cylinders.

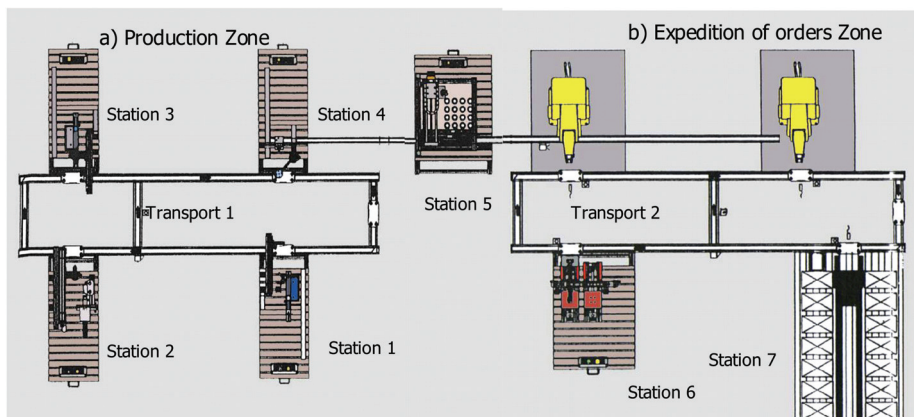


Fig. 4. Flexible manufacturing cell.

- Station 3: attaches the Cap.
- Station 4: checks the components and acts as a link between the production zone and storage area for all the finished components.

Transport 2 (right hand side of Fig. 4) arranges the transport of the cart in the expedition of orders zone between stations 6, 7, 8 and 9 that do the following operations:

- Station 6: is in charge of retrieving the base (black or white) where the three cylinders will be situated.
- Station 7: is a servo controlled storage area that stores the orders in one of its eighty positions.
- Station 8: will grasp the stored cylinders in the station 5 and inserts them in the base according to the order introduced by the user.
- Station 9: removes orders from the cell.

Station 5 links the production and expedition zones, acting as or intermediate storage area of 16 positions. The different pieces (cylinders) are dispatched depending on the production orders and the storage policy of station 5.

The original control system of the cell consisted of a programmable logic controller (PLC) that controlled each station. A real-time industrial net connected all of the PLCs. We then installed a new control system based on a fieldbus. The fieldbus was installed at stations 1, 2, 3, and 4, and at Transport 1 (see Fig. 5). Stations 1 and 4 have Inline modules (Phoenix>Contact 2006), and stations 2, 3 and Transport 1 have Advantys STB modules (Schneider_Electric. 2006).

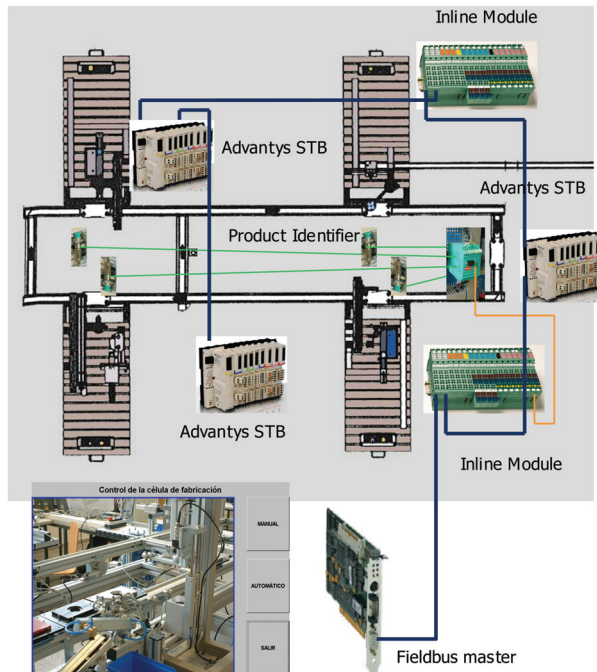


Fig. 5. Fieldbus and Product Identifier

Both Transport 1 and Transport 2 identify the contents of every pallet by means of the read and write heads of the memories attached to the pallets.

At the beginning of an operation at a cell station, to determine the operation that has to be performed, the memory of the pallet that arrives must be read. At the end of the manufacturing operation, the memory of the pallet must be updated.

The product identifier module is a resource shared by all of the stations. Communication with the identifier module is achieved using a serial port inserted in the Inline module of Station 1. Access to the module must be protected from concurrent access.

5. The input/output module

Communication with the control system could be established with the coordinators directly accessing the communications network to read or write each time that they need to interact

with the cell. This strategy would not be very efficient because even writing or reading only one station variable would involve a complete reading or writing of the bus. This task has therefore been left to the communications layer. The coordinators access the data they need through monitors that guarantee mutual exclusion in accessing variables. In this way the implementation of the control layer is independent of the system used to communicate physically with the cell.

The communication between the control module and the I/O module takes place via the station monitors. The control module reads the input values from the station monitors, executes the PLC program, and writes the output values in the station monitors. The I/O module reads the input signals of the fieldbus and leaves them in the station monitors. It then collects the output values and sends them via the fieldbus to actuators.

The execution of the periodic task of communication with the fieldbuses is run in a real-time thread at a lower priority than the task of executing the PLC program.

We have been developed classes to allow communication with the fieldbuses Interbus, CANopen and Modbus TCP/IP. The communication with the Interbus and CANopen fieldbuses is carried out through libraries in C++. The call to these libraries is made through JNI (Java Native Interface). Java allows fragments of native code to be incorporated in its programs that is code compiled for a specific platform, generally written in C/C++. JNI is the tool Java has to make use of methods run in other programming languages. For this reason it uses the JNI.

Interbus is a network of distributed sensors and actuators for manufacturing systems and control processes. It is an open system with advanced features and a ring topology. The basic concept of an open bus is to provide information exchange between devices produced by different manufacturers. The information exchanged includes processing data (inputs / outputs) and parameters (configuration data, programs, monitoring data). The information format is defined by means of a standard profile for the devices. Interbus has standard profiles for servomotors, encoders, robot controls, positioning controls, control and operation panels, digital inputs / outputs, analogue inputs / outputs, thermocouples, meters, frequency variators, robots, soldering control, identification systems, etc. The INTERBUS protocol, DIN 19258, is the standard interface for these profiles. This is an open standard for E/S networks in industrial applications.

A task has been developed for communicating with Interbus. This is implemented as a periodic task responsible for reading all the input variables and writing the output variables on the bus. This task is run every 10 ms, which is sufficient given the dynamics of the controlled system. The program uses the driver (native functions written in code C accessed through JNI - Java Native Interface) offered by the manufacturer of the bus PCI master card. In fact the reading and writing is not done directly on the bus but on an image of the memory of this card.

Communication with the CANopen bus is carried out through a periodic task responsible for reading the bus inputs and writing the bus output variables and, as with Interbus, this task run every 10 ms.

Communication with the Modbus TCP/IP protocol in ethernet is made directly in Java, providing communication with the input / output modules. The classes have functions for reading and writing the input/outputs of the modules using the ModBus TCP/IP protocol. It is possible to change the communication interface of each fieldbus module. Interbus, CANopen, and Industrial Ethernet are supported. If Interbus is used, the bus master card will be the Hilscher CIF50-IBM (Hilscher 2007). If CANopen is used, the master card will be

the CIF50-COM card and if Industrial Ethernet is chosen, the Ethernet card of the PC acts as the master. The Interbus topology is a ring and inputs cannot be read or outputs written individually. The reading of inputs and the writing of outputs are managed by the bus master in the same operation. Each station of the cell has a reading/writing head of the pallet memory that is connected to an identifying module of the products (Pepperl&Fuchs IVI-KHD2-4HRX) (Pepperl&Fuchs 2006)

Finally, a program also has to be developed for providing communication with the product identifier and thus be able to control production. Sun supplies a Java communications library for applications requiring communication with a device through a serial port (Sun 2006).

6. Control architecture.

One of the main objectives of this work is to define a control architecture. From a hierarchical standpoint, our proposal is about the coordination and the local control layers of flexible manufacturing systems. Rather than distribute the local control of each subsystem of a flexible manufacturing system (manufacturing cells, transports, stores) in various PLCs, we have opted to centralize them in a computer. The system inputs/outputs are distributed over several modules connected by a field bus (see Section 5, above). This approach permits an easy way of developing and debugging new control techniques.

The design and implementation of the local control of subsystems are maintained separately. In this way, separate threads running on the central computer control each subsystem. Another thread is responsible for the coordination function of the cell. Synchronization with the controlled system is achieved using an image memory of the inputs and outputs of the subsystems in the control computer, which is periodically updated through the field bus. That is, in each period, the inputs are read from the bus and the outputs are written to the bus. The cell coordinator updates the memory image. Several monitors protect the memory image because the input-output variables can be shared by more than one local controller, the cell coordinator, and some other threads, such as the Human Machine Interface HMI. Each local controller has its own monitor that protects the variables of the controlled subsystem. Fig. 6 shows the proposed software control architecture.

To synchronize the local controller's execution with the reading/writing of data in the field bus, a semaphore for each controller is used. The local controllers are cyclical but, at the beginning of each cycle, they must wait for a signal from the cell coordinator. It is periodic and, in each period, it sends a signal to all of the semaphores, which permits the execution of a new cycle of all of the controllers. Thus, the coordinator and the local controllers are periodic and have the same period. To implement this schema, the following three levels of fixed priorities are needed:

- High priority. This is reserved for the cell coordinator.
- Medium priority. All of the local controllers have the same priority.
- Low priority. This level is associated with threads that do not have real-time requirements, e.g., the HMI.

The proposed concurrent structure and priorities guarantee that the controllers always execute using updated input data and allow real-time analysis of the thread set. Following a rate monotonic approach, all of the local controller's threads run within their period (the control period) if:

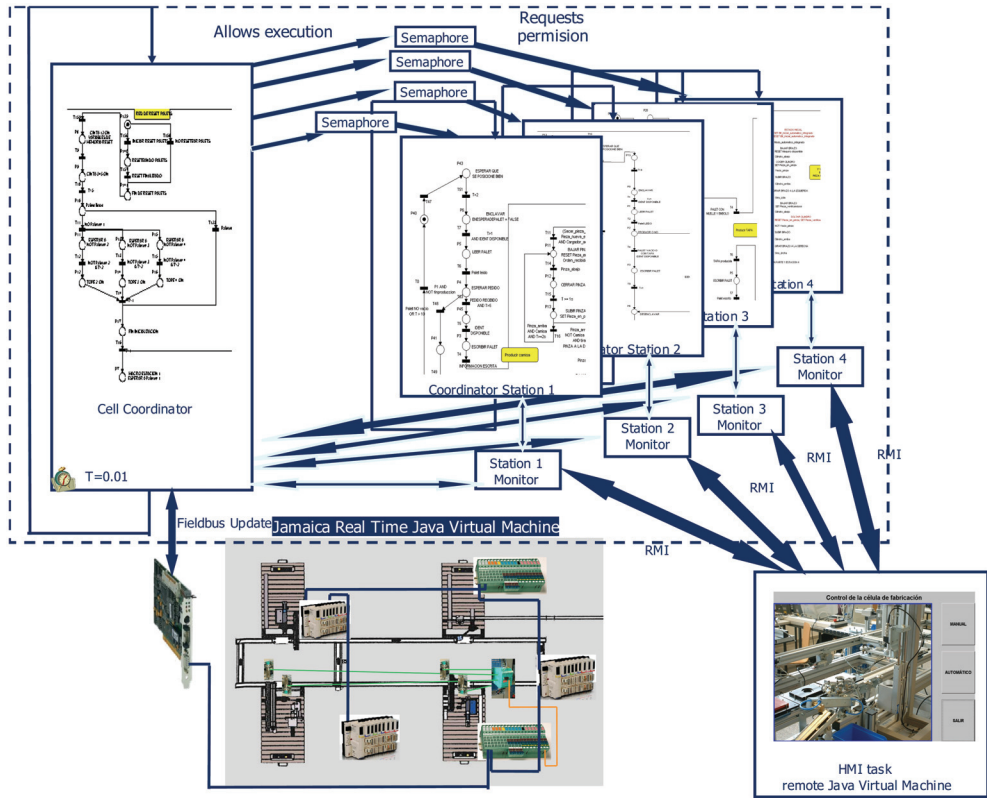


Fig. 6. Control architecture

$$C_{coord} + \sum_i C_i + b \leq T \tag{1}$$

where T is the control period, C_i is the Worst Case Execution Time (WCET) of control thread i , C_{coord} is the WCET of the cell coordinator, and b is the blocking time of the monitors. We have used the immediate ceiling priority protocol, so the blocking time is the largest WCET of all of the services provided by the monitors and called by the lower priority threads, such as the HMI. The expression imposes a restriction on the number and complexity of local controllers running on the computer and on the refresh time of the bus. If the previous condition is fulfilled, the worst-case response time for events in the system can be calculated as:

$$t_r \leq 2T + t_{bus} \tag{2}$$

That is, the response time (t_r) has a bound related to the control period (T) and the reading/writing time of the bus (t_{bus}). An example of the system response time to an incoming event is presented in Fig. 7.

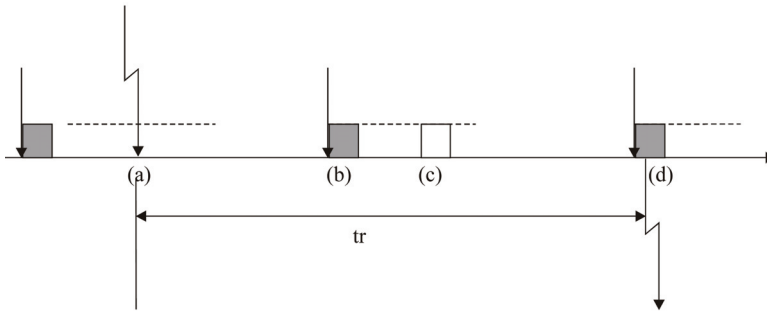


Fig. 7. System Control Time response:

- An event happens in the system (e.g., a pallet arrives at a station).
- The event is copied to the memory image of the control system.
- A local controller reads the event and establishes the reaction as changes on the memory image of outputs.
- The outputs are established in the system through the fieldbus.

In our application, the control period is 10 ms, sufficient for the dynamics of the controlled system. With Interbus, the read-write time is less than 2 ms; therefore, the response time of the real-time control will be:

$$t_r \leq 2 * T + t_{bus} = 2 * 0.010 + 0.002 = 0.022 \text{ s} \quad (3)$$

In the proposed architecture, only one thread accesses to the communication field bus and establish the refresh frequency; therefore, it is extremely simple to adapt the control application to field buses other than Interbus. At present, it is possible to execute the control over the Interbus, CANopen, and Industrial Ethernet field buses.

In this control architecture, we propose the following:

- The specification of control systems using PN and SFC that allows simulation and system analysis, and the automatic generation of code (see Section 8).
- The use of the JamaicaVM platform to support the concurrence and real-time execution of control programs.
- In addition, we have implemented a graphical task (HMI) that allows real-time visualization of the state of the control system and the PN in Execution (see Fig. 8). In Java Real Time, the graphical task is executed in a remote virtual machine and the communication is made using the *Java Remote Method Invocation System* (RMI) (Sun 2008). In a monitor, the graphical task writes the orders that the operator sends to the control system.

Those aspects are presented below.

8. The Java implementation

Previous research on deriving Java code from PN specifications (see for example (Conway, Li et al. 2002) or (Buchs, Chachkov et al. 2003)) has focused on prototyping and simulation. Our objective is to generate Java code for Real Time control systems and, thus, we have chosen to adapt classic techniques of PN implementation developed for obtaining efficient and predictable control applications.

Java is an object oriented language. As in any Java application, the code is encapsulated in a series of classes that contain a specific functionality. A set of classes that carry out related tasks are usually grouped in the same package. In the application design it has been attempted to take advantage of the opportunities offered by Java to make a modular and robust application so that future modifications can be made quickly and efficiently.

The packages of the classes implemented are described below. The basic organization is as follows:

- Petri net: a package of classes representing places, transitions and structure of a Petri net.
- SFC: a class that extends the Petri net class and allows the SFC representation.
- Applications: the application package contains the final applications of the project and constitutes the definitive set of classes both for both manual and automatic control of the cell and small manual and automatic control applications of the stations. The main content comprises the programs for the server of the applications and the interfaces required for remote communication with the RMI.
- Input/Output: contains the classes enabling communication with the controlled system through Interbus, CANopen and Ethernet.
- Control: contains the basic classes for the execution of the Petri nets. The execution algorithms are implemented in this package for the Petri nets and SFCs: Enabled Transitions, Representing Places, Brute Force, Deferred Transit and Immediate Transit Evolution Model.
- Centralized control: implements the centralized execution technique of the Petri nets. Among other classes, it contains the coordinator that executes the global Petri net of the cell.
- Decentralized control: implements the decentralized technique for the Petri net control. Among other classes, it contains the coordinators responsible for executing independently and concurrently each of the Petri nets that make up the cell system.
- Stations: monitors that contain the values of the sensors and actuators of each station.
- Serial Port: contains the classes that manage the communication with the product identifier through the PC serial port.

The first steps in the Java implementation were to develop the basic classes that allow representation of a Petri net. In addition, classes were developed that allow connections between the physical environment and the classes responsible for the execution of the Petri net, as well as classes that allow communication between different threads (monitors).

In the implementations developed in the present work, the program loads the net structure from a text file generated by a Petri net editor. Thus, the implementation is independent of the net and, therefore, is an *interpreted* implementation.

In the execution of a Petri net, a thread, the coordinator, makes the firing of transitions and updates the marking, this being a *centralized* approach. Following the approach presented in the previous section, we propose a new PN implementation technique called concurrent coordinators, in which centralized and decentralized ideas are merged. The concept is very simple; the net is decomposed into several subnets following, for example, a functional criterion, and a different coordinator implements each subnet in a centralized way. The communication between the different subnets is made using communication places that are implemented by monitors that use synchronized methods for marking and unmarking.

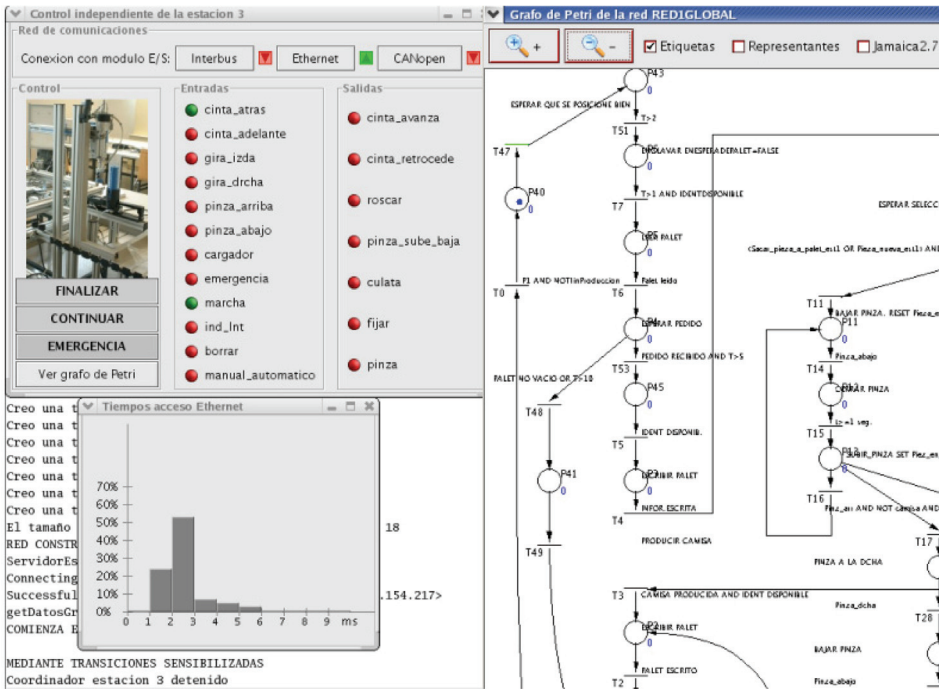


Fig. 8. Human Machine Interface.

In our practical application, the decomposition is straightforward; there are separate subnets for each station. Thus, the local controllers are coordinators that implement these subnets.

For an efficient search of fireable transitions and the update of the data structure that stores them, diverse techniques have been proposed; e.g., enabled transitions, representing and dynamical representing places (Silva 1985) (Colom, Silva et al. 1986) (Villarrol 1990). Also, for the implementation of SFCs: Immediate Transit Evolution Model (ITEVM) (Hellgren, Fabian et al. 2005) algorithm and the Deferred Transit Evolution Model (DTEVM) (Hellgren, Fabian et al. 2005). In the present work, these five techniques were implemented.

As indicated above, the cell coordinator and the local controllers are concurrent threads with real-time requirements. From the perspective of the Real Time Specification of Java, they are *RealtimeThreads*. Thus, they are scheduled following a static priorities policy with preemption. The cell coordinator is defined as periodic. The Java scheduler is responsible for the periodic activation of that thread. Fig. 9 shows the class hierarchy of our practical application.

The class *coordinator* inherits from the *RealtimeThread* class of real time Java. Each implementation technique opens a branch in the hierarchy. In the abstract class *Enabledtransitions*, the enabled transitions technique (also called transition driven) is implemented. In the abstract class *representingplaces*, the methods of the technique representing places are implemented. *Dinamicrepresentingplaces* and *staticrepresentingplaces* inherit from the previous one.

The cell coordinator and the local controllers are instances of descendants of those abstract classes. For example, Fig. 9 shows the classes in the control of the manufacturing cell that are

implemented by the enabled transitions method. All of the classes inherit the real-time characteristics.

- java.lang.Thread (implements java.lang.Runnable)
 - javax.realtime.RealtimeThread (implements javax.realtime.Schedulable)
 - control.Coordinator
 - control.EnabledTransitions
 - control.CoordinatorCell
 - control.CoordinatorEst1
 - control.CoordinatorEst2
 - control.CoordinatorEst3
 - control.CoordinatorEst4
 - control.RepresentingPlaces
 - control.StaticRepresentingPlaces
 - control.DynamicrepresentingPlaces
 - control.DTEVM
 - control.ITEVM

Fig. 9. Coordinator Class Hierarchy.

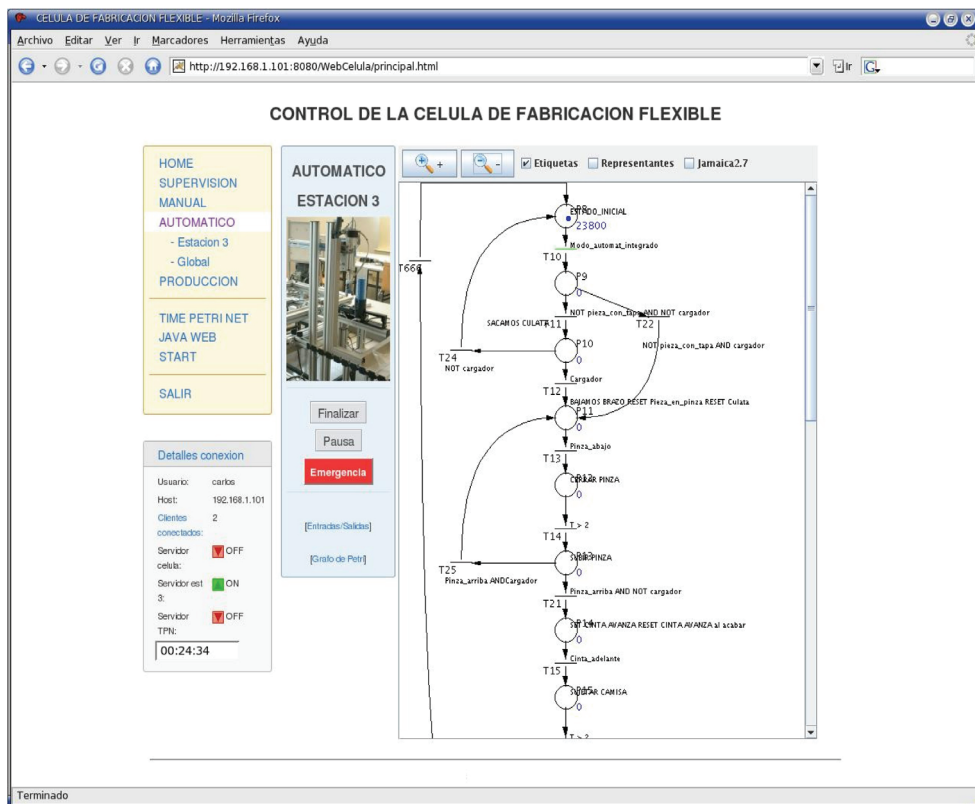


Fig. 10. Petri Net Visualization in a web browser.

In the coordinator class, we have defined the abstract methods related to the Petri net interpretation:

```
protected abstract void ContinuousAction();
protected abstract void Postaction();
protected abstract void Preaction();
Protected abstract boolean fireCondition ();
```

These methods are application-dependent and must be implemented by the developer. Also we have defined the methods for the implementation of SFCs actions. Table 1 shows the actions that can be programmed in a SFC. In a PLC cycle, the following must be executed:

- Actions which depend on the state of a step: action qualifiers N, L, D, P, P0, P1.
- In the step where is programmed the storage of the stored actions (S, SL, SD, DS) and their cancellation (R).
- The stored actions (S, SL, SD, DS)

The state of the action depends on the action flag Q and the activation flag A. The activation and action flags are activated when the action is activated; the activation flag also remains active in the cycle that turns off the action flag. The action types and qualifiers are the standard ones of the IEC 61131 (ISO/IEC 2001).

Qualifier	Description	#cycles set	
		Q	A
N	Non-stored, executes while step is active.	≥ 1	≥ 2
L	Limited, executes only a limited time while step is active.	≥ 1	≥ 2
D	Delayed, starts executing after the step has been active.	≥ 1	≥ 2
S	Stored, starts executing when the step is activated until reset.	≥ 1	≥ 2
R	Reset stored action.		
SL	Stored and limited	≥ 1	≥ 2
SD	Stored and delayed	≥ 1	≥ 2
DS	Delayed and stored	≥ 1	≥ 2
P	Pulse, executes when the step is activated.	1	2
P1	Pulse, positive flank, executes once when the step is activated.	0	1
P0	Pulse, negative flank, executes once when the step is deactivated.	0	1

Table 1. SFC action qualifiers.

The methods for the implementation of SFCs actions are described in (Piedrafita & Villarroel 2008).

The application of control consists of three separate modules. In the abstract coordinators, the control algorithms are implemented. In the non-abstract descending coordinators, the actions of the places and the conditions of firing of the transitions are implemented. The last module is responsible for the input/output. To communicate with Interbus and CANopen, the C libraries provided by the manufacturer were used. For their use in Java, several functions in JNI were developed. In the case of Ethernet, the communication was developed entirely in Java.

We used the JamaicaVM for our implementation. JamaicaVM's real-time garbage collector does not interrupt application threads; therefore, RealtimeThreads do not have to run in special memory areas, such as LTMemory or ImmortalMemory, which are not under the control of the garbage collector. The garbage collector is pre-empted by any real-time thread.

9. Development platform

We have set up a development environment for the control architecture based on PN and the Java language (see Fig. 11). The basic Java classes were extended to allow the graphical representation and animation of PN that control the system in real time. To describe the PN, we used the *Petri Net Markup Language* (PNML) (Billington, Christensen et al. 2003), which, among other advances, contains graphical information of the net elements (its graphical coordinates) thus allowing the graphical representation of PN and its evolution.

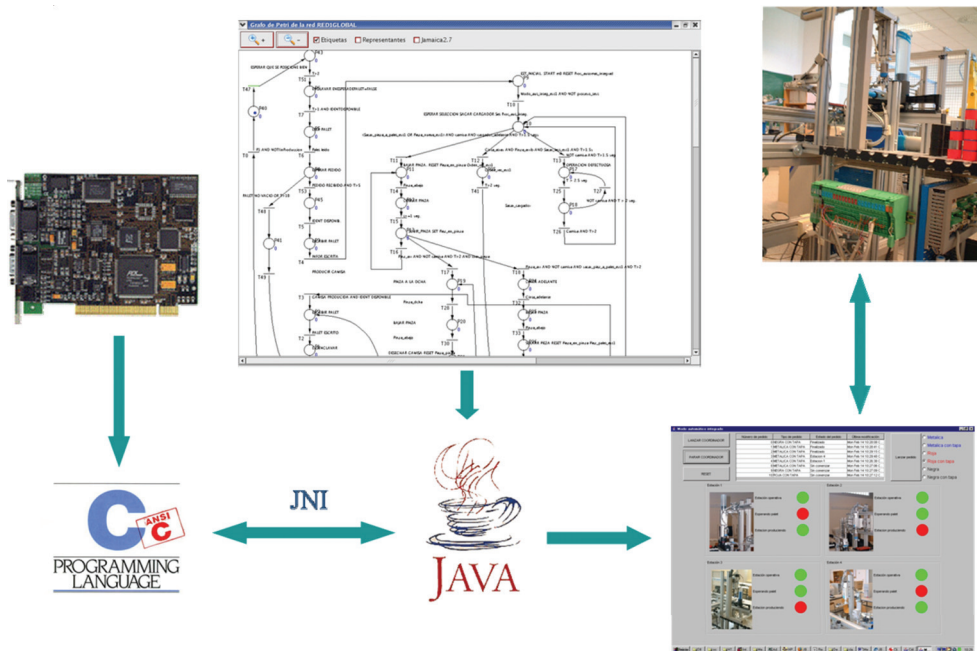


Fig. 11. Development of control applications

Petri nets were created using Pipe Editor (Bloom, Clark et al.), which allows the Petri net structure to be defined. The transitions predicates and the actions to execute are implemented directly in Java and are associated with the corresponding place and transition objects. The Petri net in PNML format is loaded by the application using parser XML, which creates a tree structure. From that tree, an object of the class Petri Net is created. Later, a thread from the coordinator class is instantiated and the Petri net object is sent as an argument.

We have developed a graphical module that allows the load, representation, and run time animation of the Petri net models. The Java Real time platform does not support the graphical classes of Java. The JamaicaVM platform has a small set of classes that allows the

drawing of points, lines and rectangles only. We decided to keep the graphics section separate from the real time control application. The communication between the two applications is performed using RMI, which allows an object that is executing in a Java virtual machine to call methods of objects or threads running on another virtual machine.

RMI applications have two separate programs: a server and a client. In our case, the server is the real time control application and the remote client is the HMI graphic interface. The full application is composed of the following (see Fig. 12):

- The real time control. In this application, the real-time execution of the Petri net occurs. It acts as a server in the RMI protocol. There is a set of class methods to allow the RMI clients to consult the state of the Petri net in execution.
- The graphical Java client. In this application, the necessary classes for the graphical visualization created in Classic Java are defined. It uses the RMI to consult the data about the PN in execution and allows the visualization.
- Web server. The Web server is programmed in Java and uses the RMI to consult the data of the real-time control.

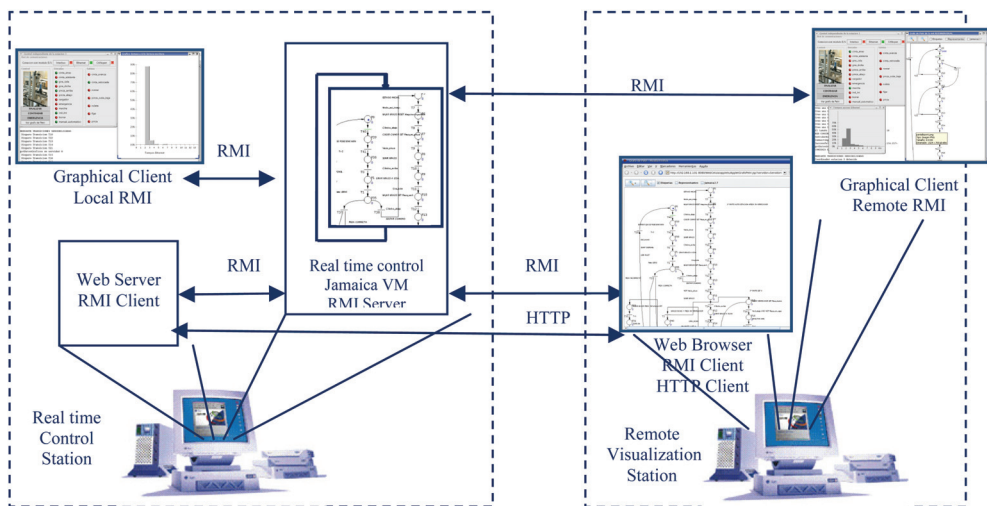


Fig. 12. Graphical interface and communication RMI

When a Web browser connects to the server, the communication is produced by means of the HTTP protocol, but when the real time visualisation of the Petri net execution applet is executed in the browser the communication with the control application takes place in the RMI protocol directly with the server.

10. Conclusions

In this paper, we have proposed a control architecture and a development environment based on Petri Nets, SFCs and the Java language. The architecture is related with the coordination and the local control layers of flexible manufacturing systems. A centralized approach permits the easy development and debugging of new control techniques. One of the mains contributions of this paper is that the proposed architecture allows the verification of the real time constraints of control systems and the use of formal tools as PNs and SFCs.

The hardware is based on a central computer that is connected to the controlled system through a fieldbus. The software architecture is based on several periodic threads. One of the threads, the cell coordinator, plays the role of the coordination layer, and the others are the local controllers of each subsystem. Synchronization with the controlled system is achieved using an image memory of inputs and outputs that is periodically updated through the fieldbus. A set of monitors protects that memory image.

The execution of the software structure over a fixed-priority scheduler allows real-time analysis of the system and a bound for the worst-case response time for events in the system can be established.

Java has been chosen as the implementation language to evaluate them in control systems. An open framework of Java classes for the PN and SFCs implementation has been developed, from the basic classes to the classes that execute the PN, and the classes that allow communication with several fieldbuses. Java lacks the real-time characteristics needed to execute the proposed architecture; therefore, the Real Time Java Specification, which provides the necessary real-time behaviour, has been adopted. The non real-time functionalities, such as the graphic interface, were implemented in classic Java because the development of graphic environments and remote web clients is easier. In this way the controlled system can be supervised remotely through the web.

The coordination function and the local controllers have been specified using Petri nets that allow simulation, systems analysis, and automatic code generation. A new PN implementation technique called concurrent coordinators, involving the use of techniques centralized in decentralized implementations has been developed. This technique is perfectly suited to the control architecture.

All of the techniques and technologies presented in this paper have been evaluated in a practical application, the control of a flexible manufacturing cell composed of manufacturing stations, transports, robots, and stores. The control system of the cell is currently running without problems. This work concludes that the Real Time Java Specification for the development of control systems based (or not) on PN is entirely valid.

The work sets the basis for more detailed research in the fields of programmed implementation of PN and SFC, languages, and execution and real-time platforms. Future work will examine the following:

- The migration to real time operative systems.
- Simultaneous control in several fieldbuses.
- Discrete event control systems decentralized and distributed implementations.

11. References

- Aicas, G. (2007). JamaicaVM Realtime Java Technology. <http://www.aicas.com/>.
- Billington, J., S. Christensen, et al. (2003). "The Petri net markup language: Concepts, technology, and tools." *Lecture Notes in Computer Science*: 483-506.
- Bloom, J., C. Clark, et al. PIPE Platform Independent Petri Net Editor, Technischer Bericht, Department of Computing, Imperial College London, 2003.
- Bollella, G. and J. Gosling (2000.). "The Real-time Specification for Java." *Computer* 33(6): 47-54.
- Briz., J. L. (1995). "Técnicas de implementación de redes de Petri. ." PhD thesis, Univ. Zaragoza.

- Buchs, D., S. Chachkov, et al. (2003). "Modelling a secure, mobile, and transactional system with CO-OPN." Application of Concurrency to System Design, 2003. Proceedings. Third International Conference on: 82-91.
- Colom, J. M., M. Silva, et al. (1986). "On software implementation of Petri nets and colored Petri nets using high-level concurrent languages." Seventh European Workshop on applications and theory of Petri nets, Oxford, July 86: 207-241.
- Conway, C., C. H. Li, et al. (2002). Pencil: A Petri net specification language for Java. Math. Dept., Macquarie Univ., Sydney, Australia.
- García, F. J. and J. L. Villarroel (1996). "Modelling and Ada Implementation of Real-Time Systems using Time Petri Nets." Proc. of the 21st IFAC/IFIP Workshop on Real-Time Programming. Gramado-RS, Brazil. November.
- Hellgren, A., M. Fabian, et al. (2005). "On the execution of sequential function charts." Control Engineering Practice 13(10): 1283-1293.
- Hilscher. (2007). "PC cards. <http://www.hilscher.com>." ISO/IEC (2001). "International standard IEC 61131-3 (2nd ed.). Programmable logic controllers—Part 3. ISO/IEC (final draft)."
- Klein, S., G. Frey, et al. (2003). PLC Programming with Signal Interpreted Petri Nets. ICATPN 2003, Eindhoven, Srpinger Verlag.
- Lewis, R. W. (1998). "Programming industrial control systems using IEC 1131-3." IEEE control engineering series 50. Open, P. L. C. (2005). XML Formats for IEC 61131-3.
- Peng, S. S. and M. C. Zhou (2004). "Ladder diagram and Petri-net-based discrete-event control design methods." Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 34(4): 523 – 531.
- Pepperl&fuchs. (2006). "IVI-KHD2-4HRX DataSheet. <http://www.pepperl-fuchs.com>." 2006.
- Phoenix_Contact. (2006). "Inline Modules. www.phoenixcontact.com."
- Piedrafita, R. and J. L. Villarroel (2006). Petri Nets and Java. Real-Time Control of a flexible manufacturing cell. Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on: 1246-1253.
- Piedrafita, R. and J. L. Villarroel (2008). Evaluation of Sequential Function Charts Execution Techniques. The Active Steps Algorithm. Emerging Technologies and Factory Automation, IEEE Conference on. Hamburg, Germany.
- Schneider_Electric. (2006). "Advantys STB. <http://www.telemecanique.com>."
- Silva, M. (1985). Las Redes de Petri en la Automatica y la Informatica. Editorial AC, Madrid, 1985, Spanish.
- Silva, M. and S. Velilla (1982). "Programmable logic controllers and Petri nets: A comparative study." IFAC/IFIP Symposium on Software for Computer Control, Madrid, Spain, October 1982: 83-88.
- Sun. (2006). "Java serial communication <http://java.sun.com/products/javacomm/>." 2006.
- Sun. (2008). "Java Remote Method Invocation. <http://java.sun.com/>."
- Uzam, M. and A. H. Jones (1996). Towards a Unified Methodology for Converting Coloured Petri Net Controllers into Ladder Logic Using TPLL: Part I - Methodology. International Workshop on Discrete Event Systems - WODES'96. Edinburgh, UK: 178 - 183.
- Villarroel, J. L. (1990). Integración Informática del Control de Sistemas Flexibles de Fabricación. Tesis Doctoral. Ingeniería Eléctrica e Informática, Universidad de Zaragoza.

Holonic Robot Control for Job Shop Assembly by Dynamic Simulation

Theodor Borangiu, Silviu Raileanu, Andrei Rosu and Mihai Parlea
*University Politehnica of Bucharest
Romania*

1. Introduction

To be competitive, manufacturing should adapt to changing conditions imposed by the market. The greater variety of products, the possible large fluctuations in demand, the shorter lifecycle of products expressed by a higher dynamics of new products, and the increased customer expectations in terms of quality and delivery time are challenges that manufacturing companies have to deal with to remain competitive. Besides these market based challenges, manufacturing firms also need to be constantly flexible, adapt to newly developed processes and technologies and to rapidly changing environmental protection regulations, support innovation and continuous development processes (Nylund et al, 2008). Although the optimization of the production process remains a key aspect in the domain of fabrication systems, adaptive production gains more and more field (Sauer, 2008). Flexible manufacturing systems should be able to quickly adapt to new situations like machine breakdown, machine recovery due to physical failure or stock depletion and also face rush orders (Borangiu et al, 2008).

In recent decades, scientific developments in the field of production control have led to new architectures including heterarchical/non-hierarchical architectures that play a prominent role in flexible manufacturing.

The **traditional** approach is mainly associated to the initial CIM concept (Computer Integrated Manufacturing) and usually leads to centralized or hierarchical control structures in which a supervisor initiates all the activities and the subordinate units respond directly in order to perform them. Due to the complexity of manufacturing problems, the usual practice has been to split the global problem into hierarchically dependent functions that operate within smaller time ranges, such as planning, scheduling, control and monitoring. This traditional approach is known to provide near optimal solutions, but only when hard assumptions are met, for example, no external (e.g., rush orders) or internal (e.g., machine breakdowns) perturbations, a priori known demands, and/or supplier reliability. Since reality is rarely so deterministic, this approach rapidly becomes inefficient when the system must deal with stochastic behaviour.

The above observations allowed researchers to design in a second approach new control architectures formed by a group of independent entities that bid for orders based on their status and future workload. There is no master-slave relationship; all the entities including the manager of a particular order are bidding for it. Due to the decentralized architecture, the entities have full local autonomy and can react promptly to any change imposed to the

system. However, because the behaviour of a production order depends on the number and characteristics of other orders, it is impossible to seek global batch optimization and the system's performance is unpredictable. These control architectures, also called emergent or **self-organized**, can be categorized in four types (Bousiba et al, 2002): *bionic & bio-inspired*, as proposed by Okino (Okino, 1993) and Dorigo & Stuzle (Dorigo et al, 2004); *multi-agent*, as proposed by Maione & Naso (Maione et al, 2003); *holonic*, as proposed by Van Brussel (Van Brussel et al, 1998); and *heterarchical*, as proposed by Trentesaux (Trentesaux et al., 1998). An analysis of the state-of-the-art has been recently published by Trentesaux (Trentesaux, 2007). His main conclusion is that the expected advantages of such architectures are related to agility: on short term such architectures are reactive and on long term they are able to adapt themselves to their environment. However, these last control architectures suffer from the lack of long-term optimality, even when the environment remains deterministic, which can be considered as a "myopic" behaviour. This is the main reason why such control architectures are not really used by industrialists at the moment.

In order to benefit from the advantages of both types of architectures, traditional and emergent, a new control paradigm was proposed by (Sallez et al., 2009) in which traditional explicit control is combined with an innovative type of control called **implicit** control. This paradigm is called *open-control*, meaning that subordinate entities are characterised by autonomy and an open communication mechanism permits them to be influenced by higher level entities directly or indirectly.

In the heterarchical control approach there is also a new research direction nowadays focused on the concept "system controlled by the product" in which dynamical information and decisional capabilities are embedded into the product, making it an active entity in the fabrication process (McFarlane et al., 2002, Zbib et al., 2008).

Rather than combining the hierarchical and heterarchical control, an approach is proposed in the current work in which the two control architectures are alternated based on the current state of the system called distributed semi-heterarchic control (Borangiu et. al, 2008). Thus, the system starts working in a hierarchical manner, using an offline schedule, in order to optimize production, but as soon as a disturbance appears it switches to a heterarchical operation mode in which resources bid for the execution of orders.

There is currently a new trend in manufacturing control to apply the principle of holons in industrial networked robotics. The interpretation of the holon as a whole particle proposes an entity which is entirely stand-alone or supreme as is (a whole), but belongs to a higher order system as a basic individual part (a particle). If a limited number of parts (holons) fail, the higher order system should still be able to proceed with its main task by diverting the lost functionalities to other holons (Ramos, 1996; Deen, 2003).

Based on Koestler's concept, the next definitions, established by the Holonic Manufacturing Systems (HMS) consortium (Van Brussel et al., 1998) were accepted and used in this project:

- *Holon*: An autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. It consists of an information- and physical-processing part. A holon can be part of another holon.
- *Autonomy*: The capability of an entity to create and control the execution of its own strategies.
- *Co-operation*: A process whereby a set of entities develops and executes mutually acceptable plans.

- *Holarchy*: A system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of holons and thereby limits their autonomy (Wyns, 1997).
- *Holonc manufacturing execution system (HMES)*: a holarchy integrating in (custom designed) software architecture the entire range of manufacturing tasks from ordering to design, and production execution.
- *Holonc attributes*: attributes of an entity that make it a Holon. The minimum set is thus autonomy and cooperativeness (Bongaerts et al., 1996, 1998; Markus et al., 1996; Morel et al., 2003).

Based on the PROSA reference architecture, several research groups have developed holonic control frameworks to operate parts of a manufacturing system (e.g. part processing on multiple machine tools, part assembling on multiple robots, etc), but only a few considered material-handling tasks (Liu et al., 1973) and transportation. The negotiation scenario, proposed by (Usher and Wang, 2000), for the cooperation between intelligent agents in manufacturing control, or the " n products on m machines" KB scheduling algorithms, proposed by (Kusiak, 1990), are limited to production planning and job scheduling, and do not consider: (a) the constraints imposed by the transportation system (e.g. cell conveyor); (b) the need to qualify (recognize, locate, check for collision-free robot access and correct robot points for part mounting) assembly components; (c) verify the assembly in different execution stages (Borangiu, 2009).

The proposed holonic control framework faces the difficulties arising when moving from control theory to practice, because: (i) the real cell conveyor is modelled, parameterized and integrated in the generic job scheduling; (ii) the material components (parts, assemblies) are described by task-dependent features which are extracted from images processed in real time for material qualifying and product inspection; (iii) the mapping of job scheduling to job execution via conveyor devices (motors, stoppers, diverters) is granted to PLC networks. In order to face resource breakdowns, the job shop production structure using networked robot controllers with multiple-LAN communication is able to replicate data for single product execution and batch production planning and tracking (Cheng et al., 2006).

The holonic implementing framework will be exemplified on a discrete, repetitive production system with part machining, robotized assembling and visual quality control capabilities. The management of changes is imposed at resource breakdown, storage depletion and occurrence of rush orders. The expected performances of the system are: high productivity (selectable cost functions: throughput, machine/robot loading, overall time), high accuracy of operations, adaptability to material flow variations and shop floor agility.

The functionalities below were imposed in the development of the holonic control system:

- adaptability and quick reaction in face of production changes (rush orders);
- real time vision-based robot guidance (GVR) during precision assembly and visual in line geometry control of products (AVI) are requirements imposed to increase the diversity and quality of services performed;
- efficient (optimal) use of available resources (robot, CNC machine tools) in normal operating mode;
- stability in face of disturbances (resource failure, storage depletion).

The paper describes: (i) the design and implementing of a PLC-based distributed control architecture for a production system with networked assembly robots and machine tools, automatically switching between hierarchical and heterarchical operating mode; (ii) the

definition of the holarchy and set up of the holon structures; (iii) the design and software implementing of operation scheduling algorithms and HMES integration; (iv) the solution adopted for fault-tolerance to robot and CNC breakdown (dynamic job reconfiguring instead of reprogramming) and high availability (redundancy in SPOF hardware and inter-device communication paths); (v) the definition and execution of part qualifying operations by real-time, high-speed image processing and feature extraction (vi) the interconnection of job-shop control processes with business processes at enterprise level, by managing offer requests, customer orders and providing feedback on the current status of batch orders.

The proposed design and implementing framework addresses a *networked robotized job shop assembly structure* composed by a number of robot-vision stations, linked by a closed-loop transportation system (conveyor). The final products result by executing a number of mounting, joining and fixing operations by one or several of the networked robots. The set of specific assembling operations is extended to on-line part conditioning (locating, tracking, qualifying, handling) and checking of relative positioning of components and geometry features. These functional extensions are supported by artificial vision merging motion control tasks (*Guiding Vision for Robots - GVR*) and quality control tasks (*Automated Visual Inspection - AVI*). Real time machine vision is used to adjust robot paths for component mounting or fastening, to check for proper geometry and pose of assembly components, and to inspect the assembly in various execution stages (Borangiu, 2004).

2. Generic holonic control model for a FMS

2.1 Description of the FMS processes

According to (Brussel et al, 1998) a fabrication system is composed of the following generic entities and domains that are associated to the production:

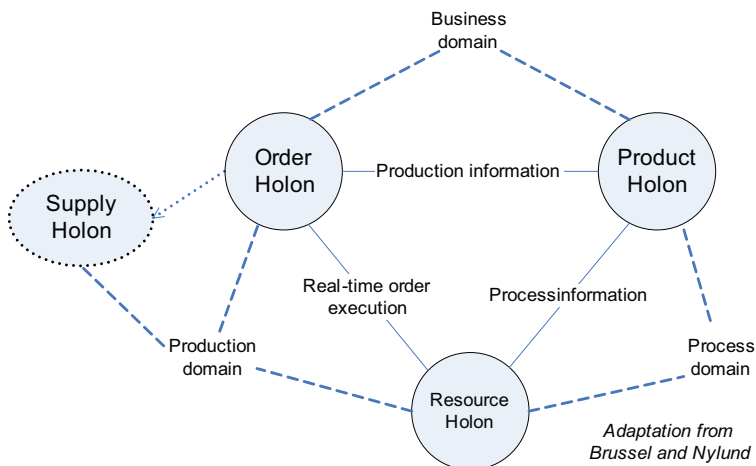


Fig. 1. HMS structure (Brussel et al, 1998, Nylund, 2008), with supply/ domain extension

Entities and domains have different purposes in the system, and are described in the PROSA reference architecture which explains the structure of a fabrication system using three basic holons: Product- (PH), Resource- (RH) and Order-Holon (OH) (Brussel et al, 1998). These entities are interconnected two by two with the process-, production- and

business domains (Nylund et al., 2008). The process- along with the production domains characterizes the system from the internal point of view. The first, *process domain*, is related with the system's capabilities to be able to execute certain operations that are needed to manufacture products offered to the clients. These capabilities are defined by the products and are realized by the resources. The second one, the *production domain*, manages the real-time information which relates the orders to the resources. This information is subject to offline and online scheduling in order to optimize the functioning of the fabrication system. The last domain, the *business domain*, relates orders to products and the fabrication system to the external world represented by clients.

Generic structure of a holon

Fig. 2 shows the structure of a generic holon, containing the digital, real, virtual and communication parts.

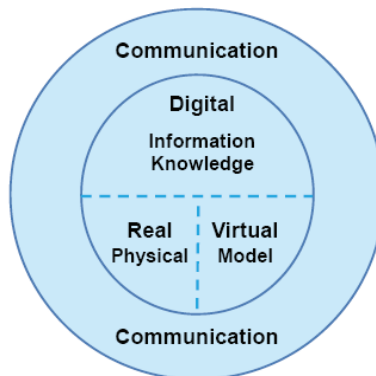


Fig. 2. The structure of a generic holon (Nylund et al., 2008)

In order for all of the different holons of the system to cooperate they must have similar structures, especially similar information structure. According to (Nylund et al., 2008) a general entity is composed of a *real part* which represents the physical resource capable of performing operations on products, a *virtual part* which is the model of the entity, a *digital part* in charge with the decision making and a communication port responsible for cooperation with the surrounding environment.

2.2 Component entities

The distributed control solution proposed in this project provides a set of functionalities rendering the material-conditioning cell flexible, rapidly reacting to changes in client's orders (batch size, type of products, alternate technologies, rush orders, updated programs), and fault-tolerant to resources getting down temporarily. In fact, the holonic control architecture proposed follows the key features of the PROSA reference architecture (Van Brussel et al, 1998; Valkaenars et al, 1994), extended with:

- Automatic switching between *hierarchical* (for efficient use of resources and global production optimization) and *heterarchical* (for agility to order changes, e.g. rush orders, and fault tolerance to resource breakdowns) production control modes.
- Automatic planning and execution of assembly component supply; automatic generation of self-supply tasks upon detecting local storage depletion,

- In line vision-based parts qualifying and quality control of products in various execution stages.
- Robotized material processing (e.g. assembling, fastening) under visual control / guidance.

As suggested by the PROSA abstract, the manufacturing system was broken down into three basic holons, the *Resource Holon* (RH), the *Product Holon* (PH), and the *Order Holon* (OH). Each of these holons may exist more than once to fully define the manufacturing cell. Order Holons are created by a Global Production Scheduler from the aggregate list of product orders (APO) generated at ERP level.

Alternate OH are automatically created in response to changes in product batches (rush orders) and to failures occurring during execution (resource breakdown, storage depletion). A holon designs a class containing data fields as well as functionality. Beside the information part, holons usually possess a physical part, like the *product_on_pallet* for OH (Duffie and Prabhu, 1994).

The way in which different types of holons communicate with each other and the type of information they exchange depends on the nature of the manufacturing cell. Fig. 3 shows the interaction diagram of the basic holon classes as they were implemented into software to solve scheduling and failure/recovery management problems. A separate software module,

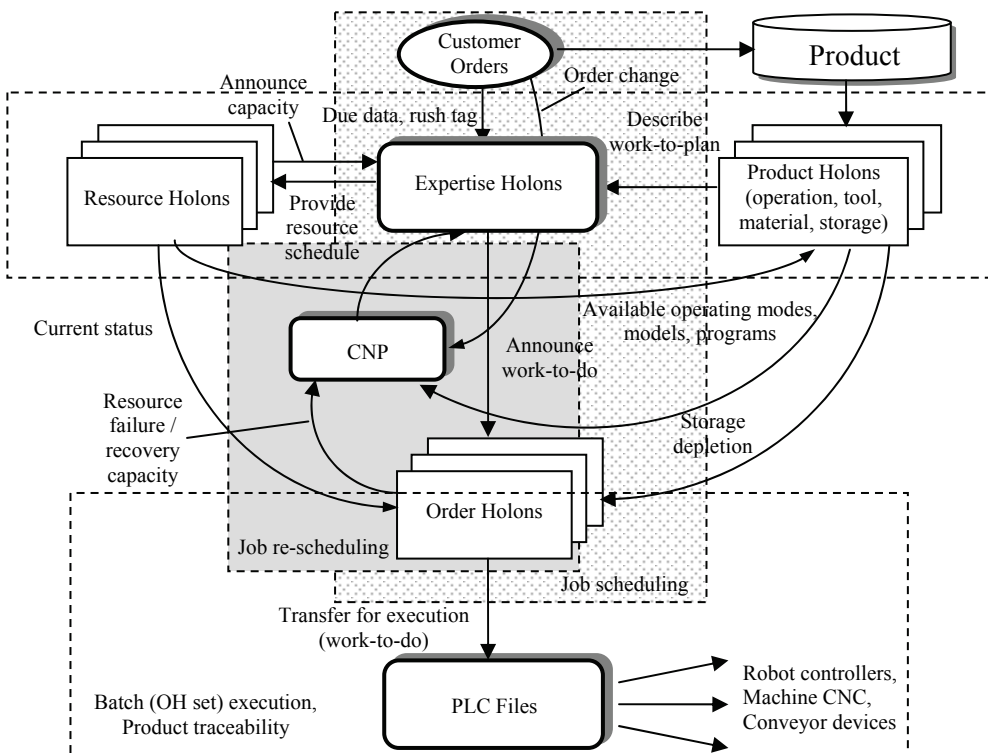


Fig. 3. Basic holon cooperation and communication structure in the semi-heterarchical control architecture

the **HolonManager** hosts all holons in form of arrays of certain types of holons and coordinates the data exchange among them.

The HolonManager entity is responsible with the planning (with help of Expertise Holons – EH) and management of OH exactly as Staff Holons in the PROSA architecture do; in addition, the HolonManager interfaces the application with the exterior (maps OH into standard PLC file and tracks OH execution for user feedback). Since a single holon may be seen as a *class object* in the object oriented programming environment (C# and .net 3.0 framework tools have been used), each of the three basic holon types was realized as a separate class.

The instances necessary to define the manufacturing cell are then hosted by the class *Holons.HolonManager*. Each type is present as an array which may be scaled dynamically if necessary. Thus, the array of *Holons.Product* class instances assumes a size of existing product types; each element represents one product type with all necessary info to generate OH of this product type.

The **resource holon** (RH) holds information about manufacturing resources (robots manipulators, grippers, machine tools, video cameras, magnetic sensors, RFID devices, a.o.). In general any resource may have a number of sub-resources (e.g. a robot manipulator with gripper with two fingers with force sensors), which are seen as holons. This project considers an entire resource with all its sub-resources as a holon without making the distinction of sub-systems. The hardware part of this type of holon is the actual physical robot and gripper with its functions. A permanent data exchange between hardware and software ensures that the actual status is accessible through the software representation of the resource holon.

A **product holon** (PH) holds information about a product type. Any (assembly) type that may be produced within the manufacturing system and resource setup must be defined in a product holon. The fact that such a holon exists does not necessarily mean that the respective product is being really assembled. Only the array of order holons (described in the next paragraph) will specify that something is manufactured and in what quantity. The product information is more of a theoretical description of the physical counter piece but not directly associated with one individual physical item, unlike the resource holon. However, the availability of assembling components is ultimately checked by the PLC prior to authorize the final transfer of a pallet carrying the product to be assembled in a robot-vision workstation (see Fig. 6).

An **order holon** represents all information necessary to produce one item of a certain product type. This holon is directly associated with the emerging item; it holds the information about the status of this very item at any time reaching from *assembly not started yet* throughout *order progressing* to *order completed*. Furthermore a complete manufacturing schedule must be computed holding all necessary information relevant for the production cell to successfully complete these orders, eventually satisfying a cost function such as the throughput or resource loading.

Before production starts for a specific aggregate order, customer commands exist in form of electronic information. If a certain product needs to be manufactured n times, n identical *raw order holons* are first created (Fig. 4).

During production execution orders can be seen as they progressively develop on a carrying support (pallet) in the system; after one order has been completed, the item gets cleared from the exiting pallet and has now a physical form. Before a schedule is defined for an

aggregate order, raw order holons are created based on the information stored in the product holon.

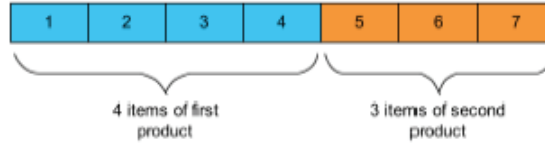


Fig. 4. Queue of two products (raw order holons) with a total of 7 items

A layer of Order Holons ($OH_p, 1 \leq p \leq P$) of variable depth, corresponding to *assembly plans* computed off line for the P final products is the output of the production scheduler fed with *raw customer orders*. A basic (quasi optimal) process plan is generated as a sequence of Order Holons (assembled products). Production planning uses the Step Scheduler developed both for production start up and resource failure and recovery situations. To formalize the OH scheduling process, the notations and definitions below are introduced:

O = Set of all operations (assembly, conditioning)

P = Set of all assemblies (final products)

OA_p = Set of operations for assembly $A_p, p \in P$

L = Set of all resource types

Q_l = Set of resources of type $l, l \in L$

t = Current scheduling time

r_{lq} = Resource q of type $l, q \in Q_l, l \in L$

For the networked assembly problem, the following types of resources were defined:

- r_{1q} = assembly robot manipulator, $q = 1, 2$: SCARA, $q = 3, 4$: vertical articulated;
- r_{2q} = gripper, $q = 1, 2$: 2(3) -finger number, $q = 3, 4$: flat / concave-contact profile;
- r_{3q} = end-effector tool, $q = 1, 2, 3$: none / bolt / screwdriver;
- r_{4q} = physical-virtual camera duality $(P_i V_j)$, $q = 1, 2, \dots, \sum nv_i, 1 \leq j \leq nv_i$, where nv_i = no. of virtual cameras defined and installed for each physical camera $i, 1 \leq i \leq 9$;
- r_{5q} = magnetic code R/W device, $q = 1, 2, 3, 4$.

Resource r_{lq} is: *operational* if it can be used after a finite time delay $\Delta_{lq}, q \in Q_l, l \in L, \Delta_{lq} \geq 0$, *available* if $\Delta_{lq} = 0$, and *down* otherwise. An *assembly plan* $AP_p^{(\delta)}$ of a product A_p is embedded in a resulting Order Holon OH as a vector of triplets, each specifying operation number o_i , processing time $t_i^{(\delta)}$ of operation o_i using assembly plan δ , and set of resources $R_i^{(\delta)}$ to process the operation o_i :

$$AP_p^{(\delta)} = [\dots, (o_i, t_i^{(\delta)}, R_i^{(\delta)}), \dots], 1 \leq i \leq f,$$

where $R_i^{(\delta)} = \{r_{1q_i}^{(\delta)}, \dots, r_{5q_i}^{(\delta)}\}$, $q \in Q_l, l \in L, 1 \leq i \leq f$.

The Step Scheduler for assembly computes off-line the $OH_p^{(\delta)}, 1 \leq p \leq P$ at batch level rendering assembly plans $AP_p^{(\delta)}$ available for products $A_p, p \in P$. One operation $o_i \in O$ in the p^{th} OH is *executable* if all resources needed to carry it out are defined as *operational* by at least one $AP_p^{(\delta)}$. Operation $o_i \in O$ is *schedulable* at time t , if

1. No other operation (mounting, inspecting) upon the same product is being processed at time t .
2. All operations preceding o_i have been completed before time t .
3. All resources needed by the basic assembly plans $AP_p^{(\delta)}$ to process operation o_i are available.

Since a single holon may be seen as a class object in the object oriented programming environment (in this project the C# and .net 3.0 framework tools have been used), each of the three basic holon types was realized as a separate class. The instances necessary to define the manufacturing cell are then hosted by the *Holons.HolonManager* class.

The array of *Holons.Product* class instances assumes a size of currently present product types; each element represents one product type with all necessary information to generate orders of this product type. The last array composed of *Holons.Order* class instances has a number of elements equal to the total count of items that needs to be manufactured. Each element defines an order of a certain product type with its specific assembly schedule.

According to the definition (Koestler, 1967) a holon is an autonomous and collaborative entity which contains a hardware and software part. In the case of the **supply holon** the hardware part is represented by the pallet carrying pieces in the system and the software is the application on the PLC which controls the path pallet and manages the exchange of messages between the workplaces and the supplied station. The relationship between the two sides is 1 to 1 and synchronization is done via the code written on the pallet (251-254). Depending on when the supplies are sent there are two types of holons: one supplying the workplaces at production start up, and the one re-supplying the workstations during production execution. The life of a Supply Holon spreads during all the manufacturing process. Unlike a normal order or a Supply Holon needed for initial feeding, where the operations to be performed are established in advance, for a Supply Holon used at re-feeding the operations are chosen dynamically depending on the usage of workstations. During production a single Supply Holon stays in the system pending for re-feeding operations. Another re-feeding constraint is the following: a pallet can carry only one type of parts for re-supply because when a workstation signals that it has an empty stock it means that it lacks a single type of piece. The type of piece is signaled to the PLC by a code similar to code that is sent when an order requires the execution of an operation. This signal is then sent to the workstation and is stored into a FIFO-type stack. From this stack re-feeding requests will be taken.

A particular case of re-feeding is the initial supply, when all workstation stocks are empty. In this case the number of Supply Holons will be extended to 4, which is the number of

pallets which ensure that the system will not block. After re-feeding only the pallet that supplies workstation during production remains in the system. The number of pallets is computed based on the configuration of the transportation system (Fig. 5).

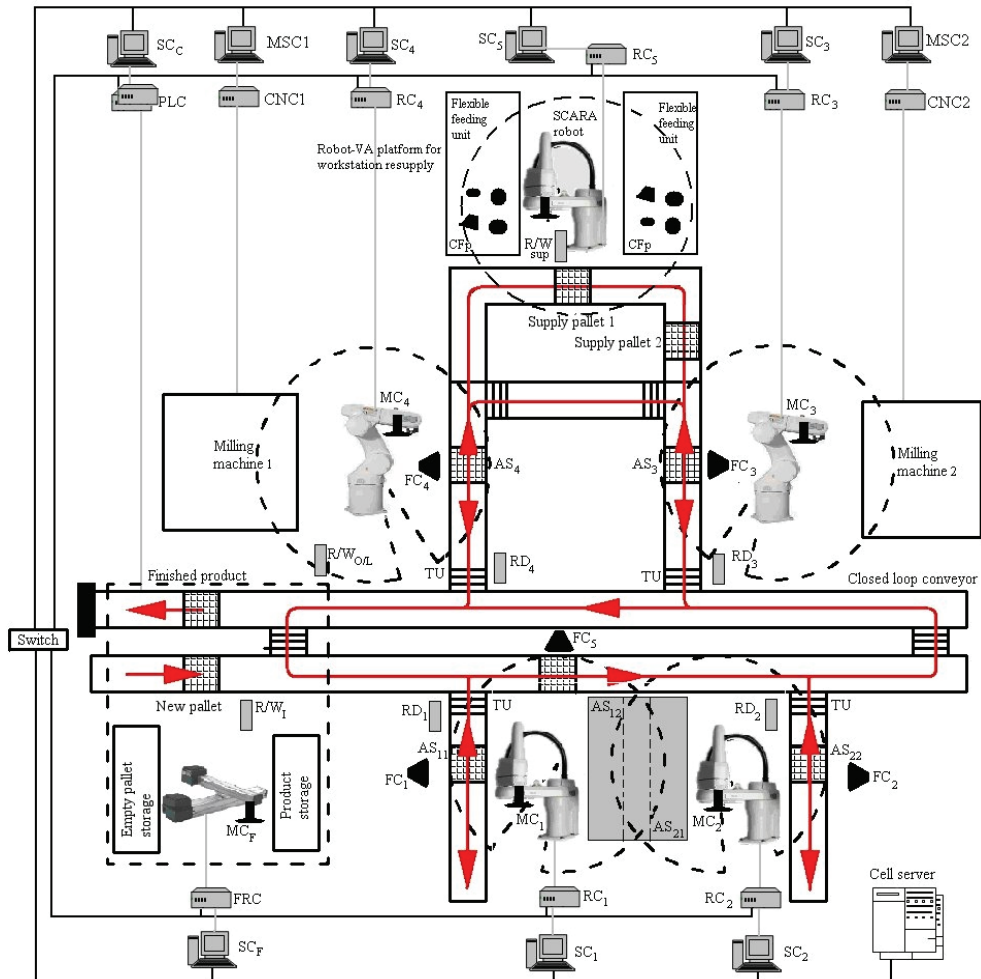


Fig. 5. Holonic manufacturing system with self-supply of assembly parts

Before production starts for a particular Aggregate List of Product Orders (APO created at ERP level), the OHs exist only in electronic format; during production execution each OH develops on a pallet in the system; after completion, the item gets cleared from the exiting pallet and has now a physical form. OHs are created from raw orders (items in the APO list) which are based on the information stored in the product holon. If a certain product needs to be manufactured n times, then n identical raw orders are created first; when OH for these

raw orders are created, the information is distinct for each OH in terms of robot stations which need to be visited and the time at which they are visited (Onori et al., 2006).

Unlike the product holon, seen as a general static entity describing a certain product type, the order holon is the actual realization of one item of a product type and undergoes many changes (of information as well as of physical nature) during manufacturing. An OH is represented by a *pallet carrier* with a unique identifier on it (magnetic tag), the *manufactured product* (on the pallet), and a *management program* running on the PLC communicating with resource controllers.

The mappings between the (holonic) system requirements and the functional architecture are included in Figs. 1 and 4. Fig. 6 describes the mappings between the functional architecture and the physical one (for a particular implementation). The real world representation refers to the model (the software counterpart of the RH, PH and OH set) of the real production system which exists at the planning level.

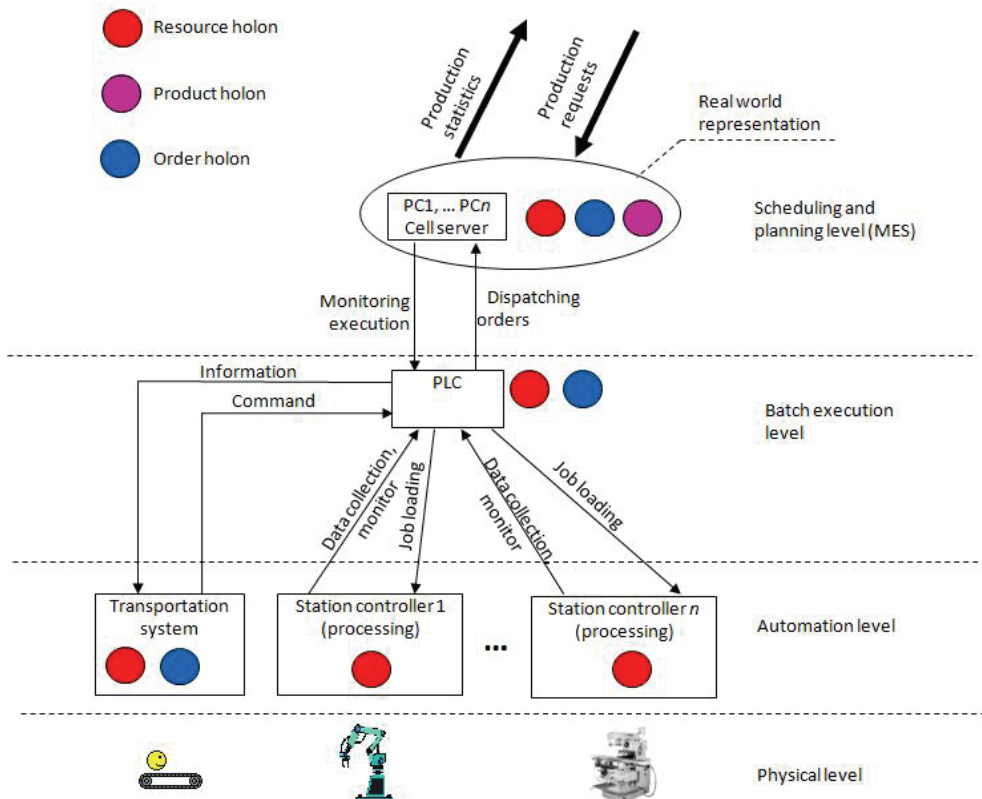


Fig. 6. Mapping between the (holonic) system and the physical architecture

3. Implementation methodology using holonic principles

The general information flow that characterizes the production system at enterprise level is described in Fig. 7:

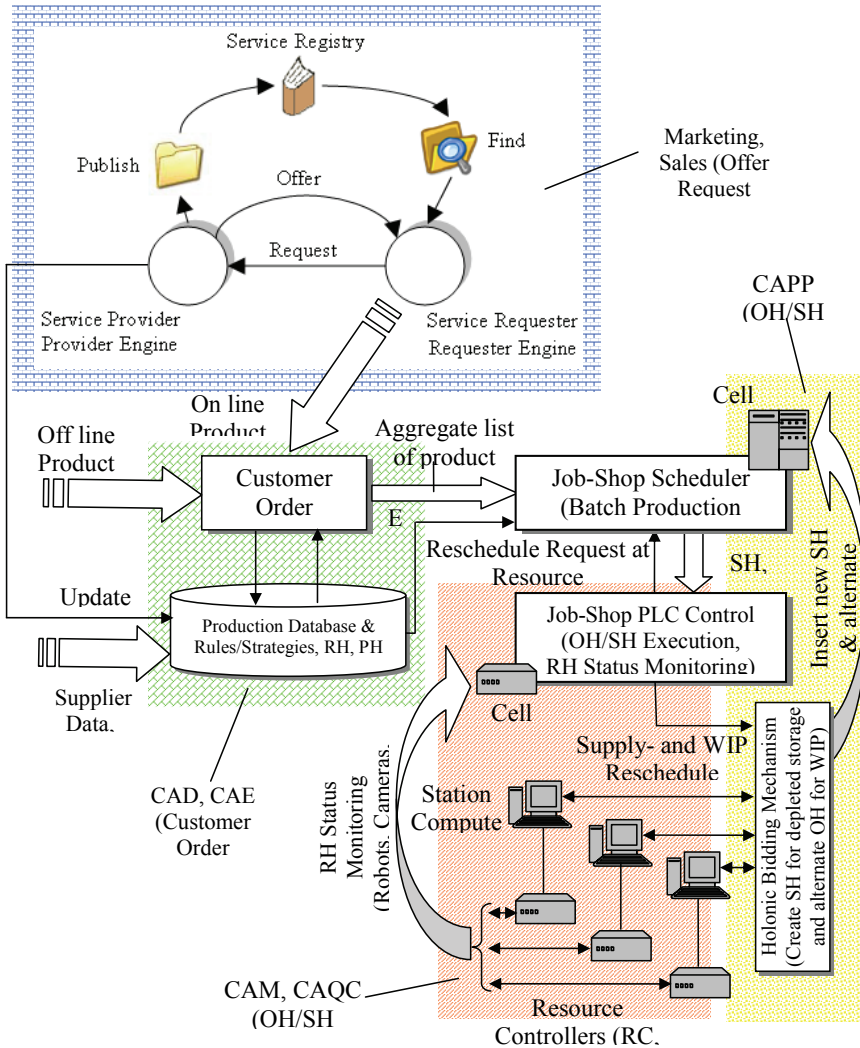


Fig. 7. Information and data flow of HMES with knowledge-based Service Oriented Architecture for customer request management

After the definition of the process and production domains the fabrication cell is ready for utilization and the business process can begin. In order for this process to be made flexible it is proposed that the information on offer requests, offers to clients, order collection and production feedback is retrieved through a web interface which is interconnected with the process as described in Fig. 8 below:

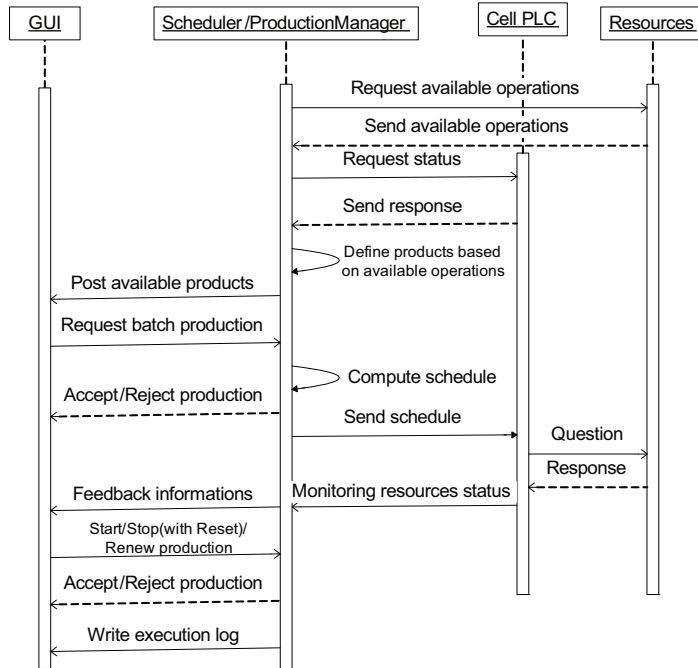


Fig. 8. Time diagram representing messages exchanged between entities from the business domain

Synchronization between the client interface and the planning and management application for the work cell is done via the exchange of text files. There will be three types of files:

- a. Input files for the scheduler
 - 1. `input_nr_orders.txt`

For a command the client will provide the manufacturer with the following details: product types, quantities and priorities. There are four levels of priority (0, 1, 2 and 3) where 3 represent the orders with the highest priority. Once this information is provided a connection between them and production domain must be created in order to report its state to the client. Therefore, besides the three fields that define a client order another field that will make contact with customer orders is added. This is the customer index.

In this way, the file has the following structure:

`nr_products$priority$product_name $index_client`

Here, `nr_orders` (from the file `input_nr_orders.txt`) is an integer which increases at each command. At each cycle of planning and production the application will retrieve the information from the file with the lowest index and then will delete the file.

- 2. `command.txt`

It is better that once the orders are introduced in production it exists a way to intervene in their execution. The reason is that for an undetermined cause the cancellation of orders' execution should be possible. Therefore, through this file that contains a single row (the

command), messages will be sent to the scheduler and management application. The commands in the file are interpreted as follows:

- start_production* - represents the start command to the scheduler
- stop_production* - stops the production immediately and cancels all orders in work.
- express_order* - represents the command that stops current production, reads a new entry (input_nr_orders file) and then plans and executes the entire batch.

b. Output files from the scheduler

1. feedback.txt

This is a file that contains the state of the orders released for execution. This file has the following line structure:

product_typedtip_producs, start_time_execution,stop_time_execution,client_index,state

having the following description: *product_type* - the type of product indicated by the representation; *start_time_execution* - the time at which the product execution has begun; *stop_time_execution* - the time at which the product execution will end; *client_index* - the index that uniquely attaches a product to a client (e.g.: if there are two clients with the same product it must be identified to whom it belongs to); *state* - represents a product state as follows: *failed_execution* (the product cannot be executed due to lack of raw materials or resources), *failed* (the product can not be executed because of a malfunction that occurred during its fabrication), *processing* (execution in progress) and *done* (product executed). This file will be analyzed with a frequency that permits sending the information to the client in real time.

At the end of the execution of a batch of products the file will contain two types of products: executed and non-executed. At every event in the system (resource failure or recovery) the scheduling is recomputed taking into account the products previously marked as compromised due to lack of resources. The non-executed products have reached this state due to two possible reasons: either they failed on the production line or there were no raw materials for their execution. For this reason the planning and management application checks one last time the "non-executed" status of products. If products still cannot be done they are finally rejected (e.g. a part is wrongly mounted because the palette doesn't arrive perfectly aligned with one of the robot's base axis; this is identified using machine vision, and the respective product remains marked as "failed").

2. lock

This is a temporary file that reflects the utilization state of the scheduler and management application: if it exists, the application is occupied with a previous order; if not, it means that orders can be sent to it. This file contains the date and time the system was blocked.

3. A response from the scheduling and management application side should exist to confirm that it is in a running/stopped state.

c. Storage files for memorizing execution logs

1. output_date_time.txt

Following the execution of a batch of products the resulting information is stored for subsequent processing. Thus there will be stored all information related to the traceability of products: operations, execution times, entry and exit times, visited resources and the final state of product (done/failed). All this information will be stored in text files whose names

start with the *output* keyword, followed by time and date when the file was written. Inside the file the following information will be found:

```
Order 1 with index 1 of type H
-----FAILED-----
Priority = ...
Insertion time: ...
Exit time: ...
Processing time: ...
Transporting time: ...
Operations:
component(on resource ..., at time ...), ...
```

The FAILED field appears only if it is the case. If production is stopped to remove / add products from execution the log file will be written at the end of the production.

2. error.txt

This file contains all the failures and recoveries the system went through in the form of records with the following structure:

```
###
```

Date: the date the failure/recovery happened

Time: the time the failure/recovery happened

Error type: string that uniquely identifies the problem and the resource that was affected.

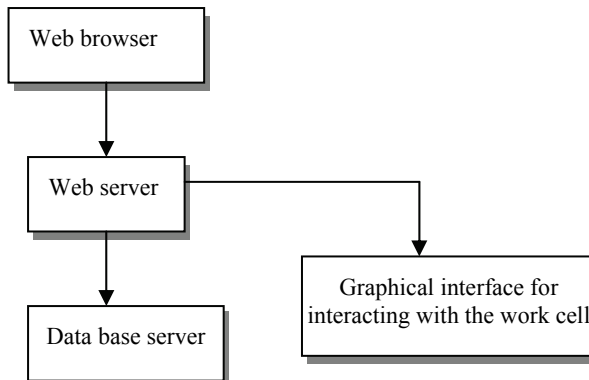


Fig. 9. Diagram of the user interaction interface

In the representation of Fig. 9, the Web browser is a common web browser like: internet explorer, firefox, opera, etc; Web server is the location that hosts the user interaction page; the Data base server is the data base server that contains the information representing client orders, products, etc.

The Graphical interface for interacting with the work cell is the module through which communication is done between the scheduling and management application and the client.

Fig. 10 presents the application modules. These are:

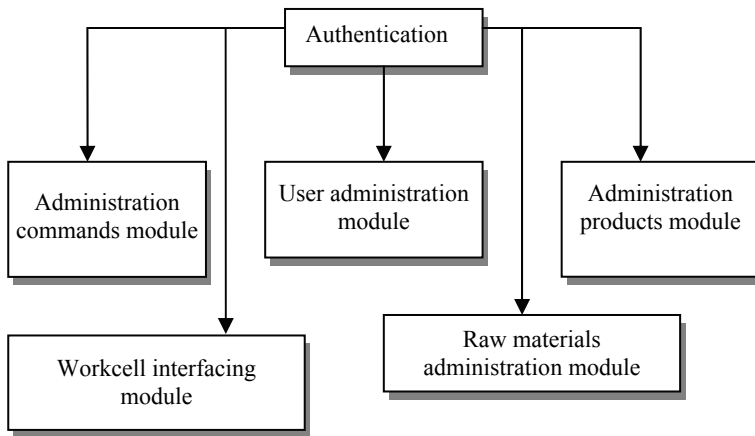


Fig. 10. Application modules

Authentication: login/logout and user administration module (for allowed zones and permissions).

Administration commands module: module that monitors the client commands currently in execution.

User administration module: module in charge of user personal data.

Administration products module: module in charge with creating a product

Administration products module: module in charge with the materials needed to create new products.

Workcell interfacing module: module needed for the communication with the application from the work cell.

Fig. 11 shows the data base structure, with the following components:

Users: a table that contains the authentication data for registered users.

User data: a table that contains user personal data address, telephone, etc.

Client orders: a table that memorizes the clients' orders already sent to execution.

Status: a table that contains the status of each client order.

Product-orders link: the link table that does the connection between the possible products to order and the orders sent to execution

Products: a table containing the list of possible products that can be executed by the system.

Products properties: a table that describes the products composition (operations to perform and precedence between them).

Products-materials link: the link table creating the connection between the list of materials and the possible products that can be executed by the system.

Materials: a table containing the existing materials that can be used to assemble products.

Materials prices: a table containing the cost of materials.

Materials properties: a table containing the description of assembly materials.

Measurement units: a table containing the measurement units for the materials used in the fabrication process.

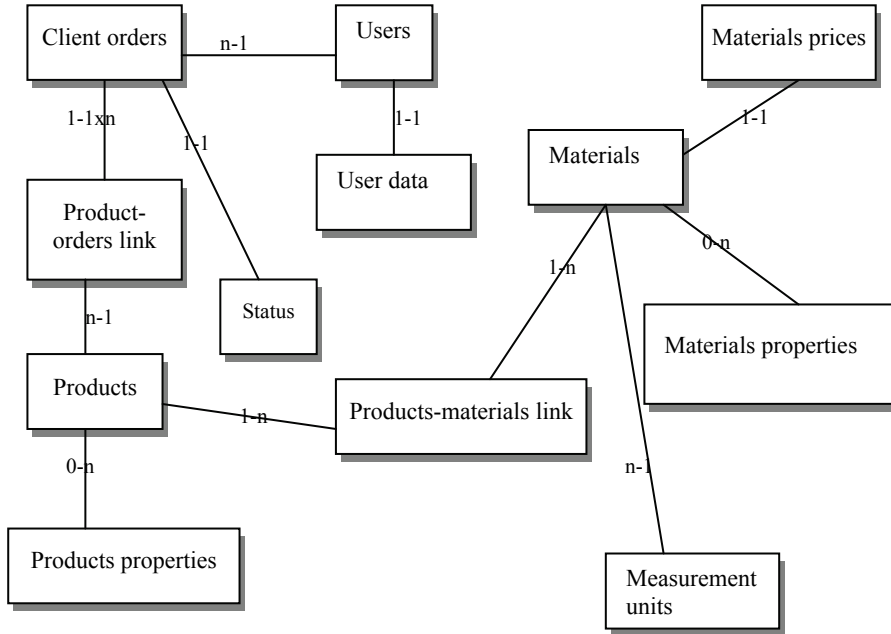


Fig. 11. Data base structure

4. Essential production processes

4.1 Step scheduler based on dynamic simulation

4.1.1 Scheduling production at batch level

Expertize Holons (EH) were defined and implemented as a set of rules creating an optimal schedule (maximizing the load of all available resources), which means that each of the four robots in the system should have a minimum of idling time. To achieve a maximum load of each robot, the conveyor system should never be jammed by any pallet carrying an item waiting to be processed by a robot. If the transport system is not blocked for most of the time each robot station will be always reachable, thus ready to receive an item and carry out a task.

Based on this idea a **Step Scheduler** has been developed. Each individual item (product) is being scheduled one step at a time. The process is initialized by generating a queue of all raw customer orders (products to be assembled - Fig. 4). Once the queue has been generated at production start up, failure detection or recovery from failure, the following algorithm is executed (an iteration of the algorithm checks and completes the following action steps):

Step 1. Check the number of pallets in the system, if there are less than 2 pallets, go to **Step 2**, else go to **Step 3**

Step 2. Choose any item randomly from the queue

Step 2.1: For the chosen item generate a list of all possible operations based on *predecessor constraints*

Step 2.2: For each possible operation find all robots capable of executing the task and calculate the waiting time for each robot before the task could be executed once the item arrives at the station

- Step 2.3:** Choose the operation with the smallest waiting time and introduce the item on a new pallet with the destination acquired before, store the current time index as insertion time
- Step 3.** Execute a step of one time index increment in the *conveyor simulator* and the robot operation execution; if a robot finishes an operation go to **Step 4**; if a pallet arrives at a workstation go to **Step 6**; else go to **Step 5**
- Step 4.** For the item that just finished an operation, store the current time index as *operation completed time*, mark the robot as *free*, then do the following:
- Step 4.1:** determine whether this item has been completed (all operations have been carried out), if so, mark the item as *completed* and send the pallet to the output, then continue with **Step 5**
- Step 4.2:** For the chosen item generate a list of all possible operations based on predecessor constraints
- Step 4.3:** For each possible operation find all robots capable of executing the task and calculate the waiting time for each robot before the task could be executed once the item arrives at the station
- Step 4.4:** Choose the operation with the smallest operation start time and send the pallet to that robot station
- Step 5.** If there are still items in the queue or pallets in the system, go back to **Step 1**, else exit the algorithm
- Step 6.** **Step 6:** For the arriving item store the current time index as operation start time, allocate this item on the robot and mark the robot as busy, continue with **Step 1**

Once an item has been introduced, it will remain in the system until it is completed. No item will leave the system and re-enter it to a later point in time. In other words any sequence (respecting the insertion criterion that the waiting time must be minimal) of alternating product types may be introduced into the system. Tests showed that different sequences (different runs of the algorithm on the exact same product definitions) yield slightly different total production times. For this reason, an alternative running mode has been integrated into the software. The so called Step Scheduler Best Sequence mode runs the algorithm 100 times and outputs the best result of these runs.

4.1.2 The Simulator scheduling tool

A Simulator has been developed and integrated in the global software system to assist and stepwise validate the creation of order holons list (i.e. the sequenced raw order holons). The simulation program routines play an essential role in the scheduling process, both at:

- Production start up, detection of a resource failure, and recovery after failure (off line, preparing production)
- Tracking of production execution, graphic monitoring (real time during production execution)

The main quality of simulating the transportation of products on pallets is the capability to vary the time base. The software furthermore uses a *transportation time matrix* which has been created by measuring the actual time used by the real system to transport a pallet from one point of interest to another (in general from one stopper or elevator to the next). The simulation's smallest time index (transport time slice) corresponds to one advancement step of a pallet and was defined as 0.5 seconds.

The transport simulation is used off-line to generate global production schedules, and in real time to track production execution. There is a fundamental difference in the use of core

routines developed to realize the correct transportation of pallets. In the case of the visual simulation, the routines run in a timed mode. This means that after each iteration of the main program loop a timer stops the program and waits until the smallest time index of 0.5 seconds has passed by and only after that allows another iteration of the main program loop. The result of this pause is the fluent running in 0.5 second steps of the simulation which, in combination with the measured transportation time matrix, corresponds exactly to the behaviour of the real system.

When these routines are used to solve the scheduling problem, they transport the pallets in the system with an infinitely high speed (limited by the computer calculation speed). As soon as an iteration of the transport functions terminates, the next one starts. Since none of the dimensions or the transportation time matrix has been changed for this kind of simulation, the resulting time indexes still correspond exactly to 0.5 seconds and may directly be used to define the production schedule.

The only difference is that time has been compressed at maximum by doing the calculations in absence of a timer which ensures the realistic execution of the simulation. The simulation functions check at each iteration all the pallets which are currently in the system. A pallet gets transported one step if the conveyor segment is running and if there is no active stopper or elevator at the pallet's present position.

Certain constraints given by the cell architecture ask for another control layer which ensures that odd situations do not occur while the system is operational. Since there are four robot stations in the cell, the number of pallets with products circulating in the system was limited to five (including the one just leaving the production system).

4.2 Failure management and system integration

The fail-safe mechanism for controlling production is triggered whenever a resource (robot controller, sensor, video camera) is down or the result of an operation is negative (visual inspection). With the help of the basic holons RH, PH, and OH and the scheduling algorithm based on EH, a *FailureManager* was created. A virtually identical counterpart, the *RecoveryManager*, exists, which takes care of the complementary event when a resource recovers from the malfunctioning and comes back online.

Alternative process plans, **triggered by resource failure/recovery, local storage depletion or occurrence of rush orders**, are automatically pipelined: (a) at the horizon of p_E products in course of execution in the system, based on heterarchical contract negotiation schemes (e.g. CNP) between valid resources, and (b) at the global horizon of $P - p_T - p_E$ remaining products, p_T = number of terminated products, based on hierarchical GSP. Two categories of changes are considered:

1. Change occurring in the resource status at shop floor level: (i) breakdown of one manufacturing resource (e.g. robot, machine tool); (ii) failure of one inspection operation (e.g. visual measurement of a component/assembly); (iii) depletion of one workstation storage (e.g. assembly parts are missing in one local robot storage).
2. Change occurring in production orders, i.e. the system receives a *rush order* as a new batch request (a new APO).

All these situations trigger a fail-safe mechanism which manages the changes, providing respectively fault-tolerance at critical events in the first category, and agility in reacting (via ERP) to high-priority batch orders. A *FailureManager* was created for managing changes in

resource status. A virtually identical counterpart, the *RecoveryManager*, takes care of the complementary event when a resource recovers from breakdown or missing parts are fed to the empty storage.

The states describing the processing capabilities of a resource and the actions taken while transiting from one state to another are presented in Fig. 12.

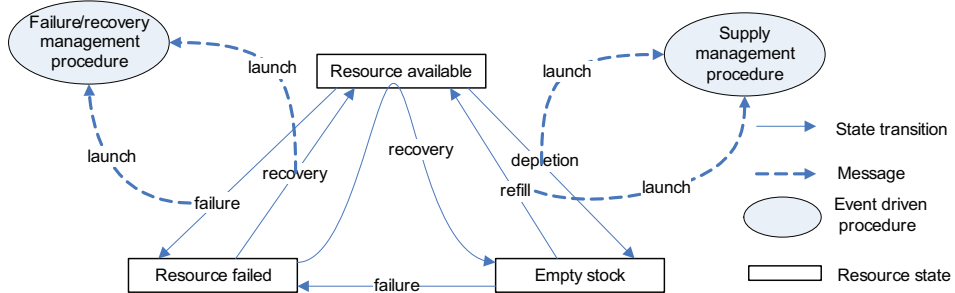


Fig. 12. Actions taken when a resource is changing from a state to another

Upon monitoring the processing resources (robots), their status may be at run time: *available* - the resource can process products; *failed* - the resource doesn't respond to the interrogation of the PLC (the entity responsible for Order Holon execution), and consequently cannot be used in production; *no stock* - similar to failed but handled differently (the resource cannot be used in production during its re-supply, but it does respond to PLC status interrogations).

There are two types of information exchanges between the PLC (master over OH execution) and the resource controllers (robot, CNC) for estimation of their status during production execution:

- *Background interrogation*: periodic polling of RQST_STATUS and ACK_STATUS digital I/O lines between the PLC - OH coordinator and the Resource Controllers (robot, CNC).
- *Ultimate interrogation*: just before taking the decision to direct a pallet (already scheduled to a robot station) to the corresponding robot workplace, a TCP/IP communication between the PLC and the robot controller takes place (see Fig. 13). This communication practically validates the execution of the current OH operation on the particular resource (robot).

In this protocol, READY is a signal generated by the Robot Controller indicating the *idle* or *busy* state of the resource (robot). The PLC requests through its digital output line RQST-JOB to use the robot for an assigned OH operation upon the product placed on the pallet waiting to enter the robot workstation. D1 details the scheduled job via the TCP PLC transmission line from the PLC to the Robot Controller. The Robot Controller indicates in D2 job acceptance or denial via the TCP Robot transmission line. When the job is accepted, the pallet is directed towards the robot's workplace, where its arrival is signalled to the Robot Controller by the Pal in Pos digital output signal of the PLC.

Job Done is a signal indicating job termination (D3 details the way the job terminated: success, failure). T1 is the decision time on job acceptance (storage evaluation etc), T2 is the transport time to move the pallet from the main conveyor loop to the robot workplace, and T3 is the time for job execution.

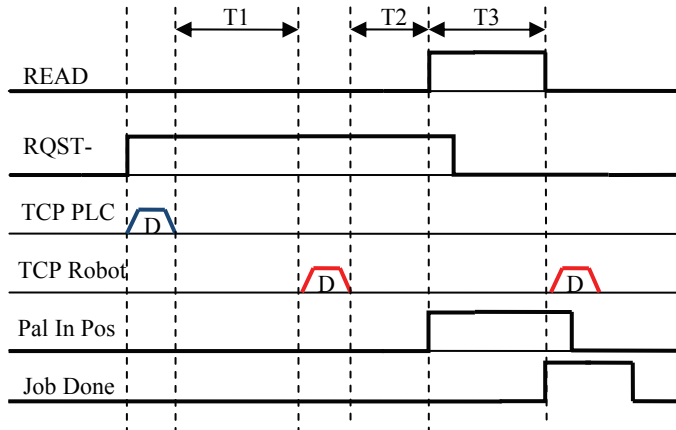


Fig. 13. Communication protocol between the PLC and a Robot Controller for authorizing an OH operation execution

Upon periodic interrogation, the entity coordinating OH execution – the PLC – checks the status of all resources, which acknowledge being *available* or *failed*. The ultimate interrogation checks only the state of one resource – the one for which a current operation of an OH was scheduled; during this exchange of information, the PLC is informed whether the resource is *available*, *failed* or *valid* yet unable to execute the requested OH operation upon the product due to components missing in its storage (*no stock* status).

When the **failure** status of a resource is detected, the *FailureManager* is called, executing a number of actions according to the procedure given below (Fig. 14):

1. Stop immediately the transitions of executing OH, i.e. the circulation of *products_on_pallets* in the cell; production continues however at the remaining valid resources (robots, machine tools).
2. Update the resource holons with the new states of all robots.
3. Read Order Holons currently in execution (which are currently in the production cell).
4. Evaluate all products if they can still be finished, by checking the status of each planned OH:
 - if the OH was in the failing robot station, mark it as failed and evacuate its *product_on_pallet*;
 - if the OH is in the system, but cannot be completed anymore because the failed resource was critical for this product, mark it as failed and evacuate its *product_on_pallet*;
 - if the OH is not yet in the system, but cannot be completed due to the failure of the resource which is critical for that product, mark it as failed (n_e is the total number of such OH).
5. For the remaining $n'_{wip} = n_{wip} - n_{fail}$ schedulable OH in the system, locate their *products_on_pallets* and initialize the transport simulation associated to the current operational configuration of the system. Authorise the n'_{wip} OH to launch Contract Net Protocol-based negotiations (HBM) with the remaining available Resource Holons for re-scheduling of their associated operations. n_{wip} are the OH currently introduced in

the system (in the present implementation, $n_{wip} \leq 5$), and n_{fail} is the total number of OH currently in the system, which cannot be finished because they need the failed resource at some moment during their execution.

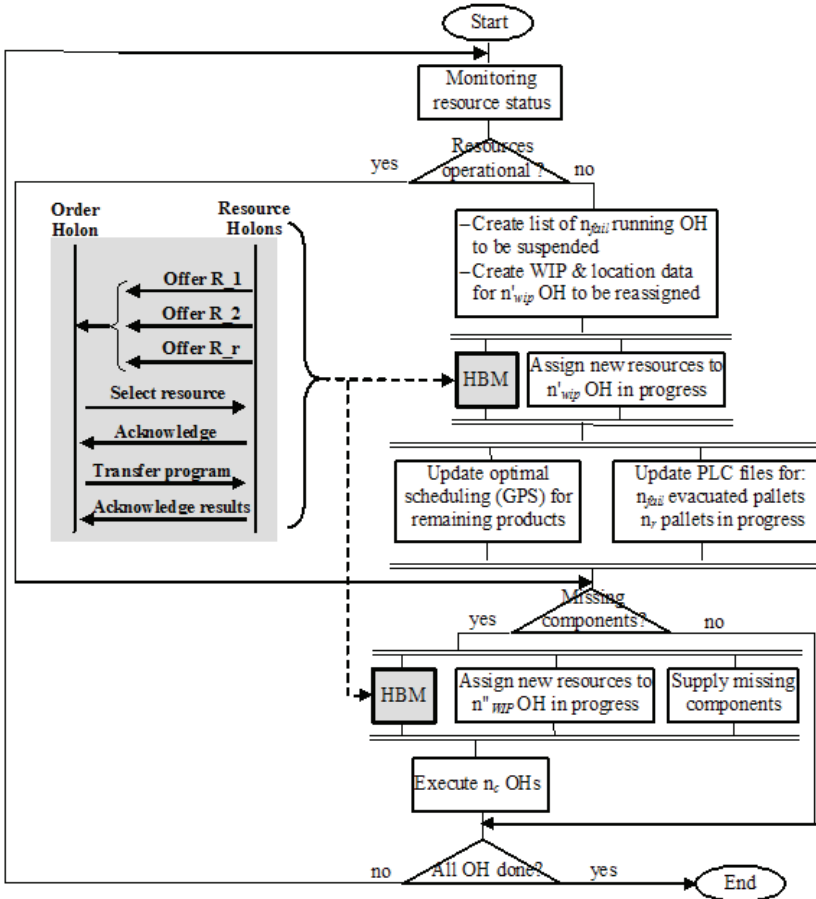


Fig. 14. Dynamic OH rescheduling at resource failure/storage depletion with embedded CNP job negotiation

6. Run the Global Production Scheduling algorithm for the $N - n_{fin} - n_{wip} - n_e$ OH not yet introduced in the system, where a number of N OH was scheduled in total and n_{fin} OH were finished.
7. Delete the orders stored on the system and transfer the updated orders to the system. Resume *product_on_pallet* transfer within the transport system (allow OH transitions in the system).

It might happen that a failed robot gets fixed before the current manufacturing cycle is finished. In this **recovery** case, the cell regained the ability to run at full capacity but the lined up orders do not make use of this fact, as they are managed by the system in a

degraded mode. The procedure of rescheduling back the Order Holons is virtually identical to the one used in case of failure; the main difference is that none of *the products_on_pallets* being currently processed need to be evacuated since there is no reason to assume they could not be completed. Any orders that were marked as failed due to resource unavailability are now untagged and included in the APO list for scheduling as they may be manufactured again due to resource recovery (Lastra and Delamerm, 2006; Leitao *et al*, 2007).

4.2 Automatic re-supply of workplaces

In case of local **storage depletion**, the OH waiting to enter the robot station with exhausted storage will be either delayed if the resource is critical or re-scheduled to another resource disposing of the missing component and able to perform the current operation. In such a situation, two actions take place:

1. One Supply Holon (SH) is created by the GSP, by specifying the type and number of parts to be retrieved, the supply source (a central cell storage tended by a SCARA robot under visual guidance), and the restoring destination (the exhausted local robot storage). The SH is immediately started.
2. From the n_{wip} OH currently in execution, n_d will be delayed until the empty storage, which is critical for certain of their mounting operations, is restored and $n_{wip}'' = n_{wip} - n_d$ OH will be re-scheduled by the holonic bidding mechanism (HBM) to robots disposing of necessary assembly parts.

A lock is put on the system, and no further OH (a new pallet) is introduced in the system until the last one of the n_d delayed OH is completed and exits the system. When both the SH and all n_d OH are terminated, the lock is suspended and the remaining OH are introduced in the system in packets of n_{wip} , their re-scheduling being not necessary.

4.2 Treatment of rapid orders

The system is agile to changes occurring in production orders too, i.e. manages **rush orders** received as *new batch requests* from the ERP level while executing an already scheduled batch production (a sequence of Order Holons).

Because of the similarities between a task run on a processor and a batch of orders executed in a cell (both are preemptive, independent of other tasks/batches, have a release, a delivery date and an fixed or limited interval in which they are processed), it was decided to use the Earliest Deadline First (EDF) procedure to schedule new batches (rush orders) for the cell.

Earliest Deadline First (EDF) is a dynamic scheduling algorithm generally used in real-time operating systems for scheduling periodic tasks on resources, e.g. processors (Sha *et al.*, 2004, Lipari, 2005). It works by assigning a unique priority to each task, the priority being inversely proportional to its absolute deadline and then placing the task in an ordered queue. Whenever a scheduling event occurs the queue will be searched for the task closest to its deadline. A *feasibility test* for the analysis of EDF scheduling was presented in (Liu and Layland, 1973); the test showed that under the following assumptions: (A1) all tasks are periodic, independent and fully preemptive; (A2) all tasks are released at the beginning of the period and have deadlines equal to their period; (A3) all tasks have a fixed computation time or a fixed upper bound which is less or equal to their period; (A4) no task can voluntarily stop itself; (A5) all overheads are assumed to be 0; (A6) there is only one

processor, and $\sum_{i=1}^n \frac{C_i}{T_i} \leq 1, n = \text{number of tasks}, C_i = \text{execution time}, T_i = \text{cycle time}$ (a set of

n periodic tasks can be scheduled if the utilization of the processor (resource) is less than 100%).

A *batch* or Aggregate Product Order list (APO) is composed of raw orders (list of products to be manufactured); this is why two different batches are independent. Nevertheless, there is a difference between a task and a batch of products: a task is periodic while a batch is generally a periodic. This means that instead of testing the feasibility of assigning batches to the production system considering the equation above, one can use the following test: "for an ordered queue (based on delivery date) of n batches with computed makespan, if

$$\sum_{j=1}^i \text{makespan}_j \leq \text{delivery_date}_i, i = \overline{1, n}, \text{ then the batches can be assigned to the production}$$

cell using EDF without passing over the delivery dates".

This EDF approach is used to insert **rush orders** in a production already scheduled by the GPS; the steps below are carried out for inserting a new production batch (rush order) during the execution of a previously created sequence of Order Holons (see Fig. 15):

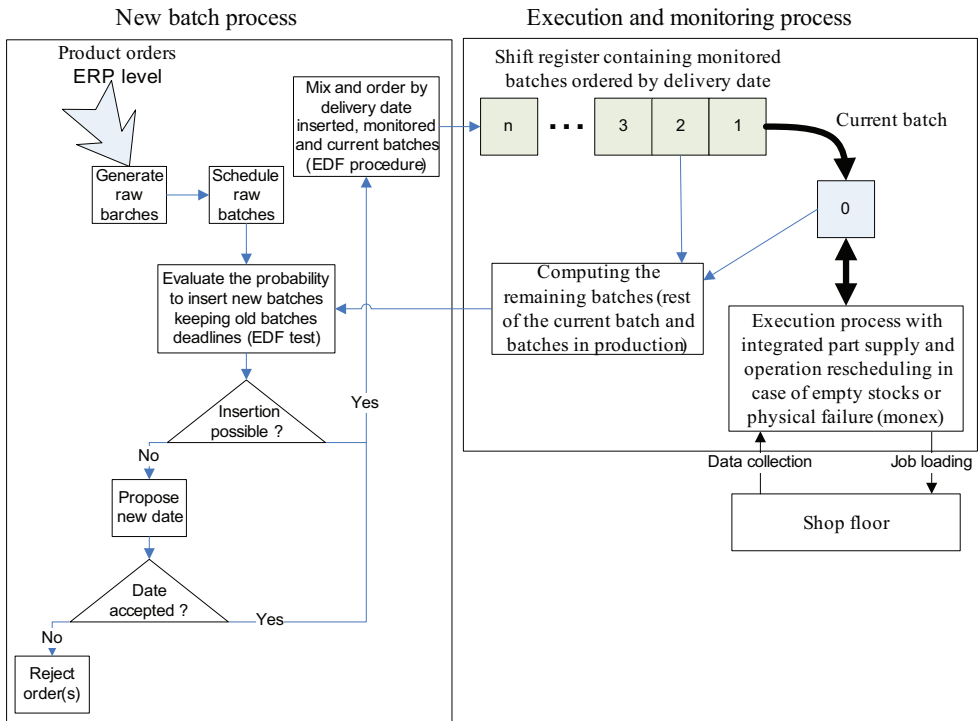


Fig. 15. Add *rush order* diagram and integration with dynamic job re-scheduling based on CNP negotiation

0. Compute the remaining time for finishing the rest of the current batch (if necessary).
1. Insert new production data: product types, quantities, delivery dates.
2. Separate products according to their delivery date.
3. Form the entities "production batches" (a *production batch* is composed of all the products having the same delivery date).
4. Generate raw orders inside the production batches (APO lists).
5. Schedule the raw orders (using a GPS algorithm, e.g. KBS or Step Scheduler), compute the makespan and test if the inserted batch can be done (the makespan is smaller than the time interval to delivery date if production starts now).
6. Analyse the possibility of allocating the batches to the manufacturing cell using the Earliest Deadline First procedure and second equation for feasibility test.
7. Allocate the batches on the real production system according to the EDF procedure.
8. Resume execution process with new scheduled Order Holons.

In this mechanism for the management of changes in production orders, an *inserted batch* is a batch that arrives while another one is in execution. A *monitored batch* is one whose orders are scheduled and assigned to the cell (it has a priority and is waiting to enter execution). A *current batch* is one in execution.

The capability of adding rush orders to production needs a new entity – the **batch**. In this way job scheduling is done at batch level (all orders with the same delivery date are scheduled together) and then batches are assigned to the cell according to their delivery date, using the EDF procedure (Table 1).

Name	Description
batch_name	Name or index of the batch
delivery_date	Delivery date of the orders
requested_products	Vector containing the products to be executed
used_resources	Vector containing the configuration used for current batch planning
orders_to_execute	Vector containing the entities OH already scheduled using a specified cell structure (defined by the variable used_resources)
makespan	The time interval needed for the current batch to be executed if started now and not interrupted (it is a result of scheduling)

Table 1. The minimal structure of a *batch holon*

Because the process of batch execution is interruptible (preemptive system), new batches (rush orders) can be introduced exactly at the moment of their arrival. The insertion process is triggered by the arrival of a "new order" event; a real-time acceptance response can be provided (via the ERP level) to the customer if the rush order can be executed by the requested delivery date.

5. Experimental results

The distributed control solution was implemented, tested and validated on a real manufacturing structure with industrial assembly robots and 4-axis CNC milling machines, using the holonic approach. This development platform was recently put in place in the **Centre of Research in Robotics and CIM** within the University Politehnica of Bucharest.

The described holonic implementation framework allows networking equipment from different producers. The cost of the development platform is directly reflected in its high precision performances, integrated inspection services, relaxation of material presenting constraints, fixture simplification and management of changes.

The control structure is fully operational, both in the normal hierarchical mode and upon switching automatically to the heterarchical one in response to rush order requests, part supply and resource failures.

An example of production definition at batch level for four products (H-, U-, L-, and C-type products) resulting from the succession 8 operations consisting of assemblies, milling and visual inspections is further analysed.

For the experiments reported, the number of products simultaneously in execution was limited to 5. Table 2 below gives the production times resulting from the Step Scheduler RSRP computation in two scenarios: (i) only H-type products; (ii) equal number of H-, U-, L- and C-type of products within one of the four batch sizes (batch sizes were 4, 20, 40 and 60 products):

Batch size	Production time [time units]		Worst recovery time [time units]	
	H-type [RSRP / CNP]	Equal number of H-, U-, L-, and C-type [RSRP/CNP negotiation]	Alternate OH at [packet = 5] level (resource <i>i</i> failure: <i>RiF</i>)	New SH for restoring Local Storage <i>i</i> (LS _{<i>i</i>}) at depletion
4	684 / 734	663 / 687	6.4 (R1F)	97 (LS1)
20	2841 / 3112	2550 / 2712	6.5 (R2F)	112 (LS2)
40	5485 / 5962	4934 / 5288	6.8 (R3F)	136 (LS3)
60	8129 / 9089	7362 / 7902	6.5 (R4F)	83 (LS4)

Table 2. Production time for H-, U-, L- and C-type batches and resuming times at resource failure

The system's behaviour was tested with good results at storage depletion (less than 68 seconds to generate a SH and restore the furthest local robot storage) and resource failure (SC, switch and RC). Future work will be directed towards integrating the process control- and ERP areas through an enhanced information management system based on RFID.

6. Conclusion

The scope of this chapter was the definition of a PLC-centred framework for developing an integrated solution aiming at controlling the resources of a flexible manufacturing system and managing of the clients' orders. The key characteristics of the proposed framework are autonomy of the control systems' resources and cooperation between them.

The general features of the proposed holonic implementing framework facilitate, beyond the product assembling with machined components, the development of any other discrete, repetitive manufacturing applications. Features like: decomposition of the production system into entities relative to the basic areas specific to an enterprise (production, process and business), description of the types of manufacturing entities and of the communication protocols that take place between them, and the decision scenarios during resource failure / recovery and stock restoring are reusable.

From the algorithmic point of view, the proposed resolved scheduling rate planner (RSRP) based on variable-timing simulation, facing the NP complexity aspect of the batch scheduling problem can be reused for any topology of the material transportation system, due to its graph-type, object-oriented description.

7. References

- Bongaerts, L., Wyns, J., Detand, J., Van Brussel, H., Valckenaers, P., 1996. Identification of manufacturing holons. Proceedings of the European Workshop for Agent-Oriented Systems in Manufacturing, Albayrak, S., Bussmann, S. (Eds.), Berlin, 57-73
- Bongaerts, P., Monostori, L., McFarlane, D., Kadar, B., 1998. Hierarchy in distributed shop floor control. Proceedings of the 1st Int. Workshop on Intelligent Manufacturing Systems IMS-EUROPE, Ed. EPFL, Lausanne, 97-113
- Borangiu, Th., 2004. Intelligent Image Processing in Robotics and Manufacturing, Romanian Academy Publishing House, Bucharest
- Borangiu, T., Ivanescu, N., Raileanu, S., Rosu, A., 2008. Vision-Guided Part Feeding in a holonic Manufacturing System with Networked Robots, Proceedings of Int. Workshop RAAD 2008, Ancona, Italy
- Borangiu Th., Gilbert P., Ivanescu N., Rosu A., 2008. Holonic Robot Control for Job Shop Assembly by Dynamic Simulation, Int. Conference MED'08, Ajaccio
- Borangiu Th., Gilbert P., Ivanescu N.A., Rosu A., 2009. An Implementing Framework for Holonic Manufacturing Control with Multiple Robot-Vision Stations, Engineering Applications of Artificial Intelligence 22 (2009), 505-521, Elsevier
- Cheng, F.-T., Chang, C.-F., Wu, S.-L., 2006. Development of Holonic Manufacturing Execution Systems, Industrial Robotics: Theory, Modelling and Control, Advanced Robotics Systems, Ed. Pro Literatur Verlag Robert Mayer-Scholz Germany, Vienna
- Deen, S.M., 2003. Agent-Based Manufacturing - Advances in Holonic Approach, Springer
- Dorigo, M., and Stuzle, T., 2004. Ant Colony Optimization. The MIT Press
- Koestler, A. The Ghost in the Machine. Hutchinson publishing Group, London, 1967
- Kusiak, A., 1990. Intelligent Manufacturing Systems, Prentice Hall, Englewood Cliffs, New York
- Lipari, G., 2005. Sistemi in tempo reale (EDF), Course Scuola Superiore, Sant'Anna, Pisa
- Liu C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of ACM 20, 1, 46-61
- Maione, G., and Naso, D., 2003. A soft computing approach for task contracting in multi-agent manufacturing control. Computers in Industry, 52, 199-219
- Markus, A., Vancza, T., Monostori, L., 1996. A market approach to holonic manufacturing. Annals of the CIRP 45, 1, 433-436
- McFarlane D, Sarma S, Chirn Jin Lung and Wong C Y, and Ashton K. 2002. "The intelligent product in manufacturing control and management". Proceedings of the 15th Triennial World Congress, Barcelona
- Morel, G., Panetto, H., Zaremba, M., Mayer, F., 2003. Manufacturing enterprise control and management system engineering: Rationales and open issues, IFAC Annual Reviews in Control
- Nylund H., Salminen, K., Andersson, P.H., 2008. A multidimensional approach to digital manufacturing systems, Proceedings of the 5th International Conference on Digital Enterprise Technology, Nantes

- Okino, N., 1993. Bionic Manufacturing System in Flexible Manufacturing System: past - present - future. J. Peklenik (ed), CIRP, Paris, 73-95
- Onori, M., Barata, J., Frey, R., 2006. Evolvable assembly systems basic principles, IT for Balanced Manufacturing Systems 220, IFIP, W. Shen (Ed.), Springer, Boston, 317-328
- Ramos, C., 1996. A holonic approach for task scheduling in manufacturing systems, Proceedings of the IEEE Int. Conf. on Robotics and Automation, Minneapolis, USA, 2511-2516
- Sallez Y., Berger T., Trentesaux D., 2009. Open-control: a new paradigm for integrated product-driven manufacturing Control, Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM '09), Moscow
- Sauer O., 2008. Automated engineering of manufacturing execution systems - a contribution to "adaptivity" in manufacturing companies, 5th International Conference on Digital Enterprise Technology, Nantes
- Sha, L. et al., 2004. Real Time Scheduling Theory: A Historical Perspective, Real-Time Systems, Vol. 28, No. 2-3, 101-155
- Trentesaux, D., Dindeleux, R. and Tahon, C., 2009. A MultiCriteria Decision Support System for Dynamic task Allocation in a Distributed Production Activity Control Structure. Computer Integrated Manufacturing, 11 (1), 3-17
- Usher, M.J., Wang, Y-C., 2000. Negotiation between intelligent agents for manufacturing control, Proc. of the EDA 2000 Conference, Orlando, Florida
- Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, L., 1998. Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry, 37 (3), 255-274
- Wyns, J., Van Ginderachter, T., Valckenaers, P., Van Brussel, H., 1997. Integration of resource allocation and process control in holonic manufacturing systems, Proceedings of the 29th CIRP Int. Seminar on Manufacturing Systems, 57-62
- Zbib, N., Raileanu, S., Sallez, Y., Berger, T. and Trentesaux, D., 2008. From Passive Products to Intelligent Products: the Augmentation Module Concept. Proceedings of the 5th International Conference on Digital Enterprise Technology, Nantes

Centralized/Decentralized Fault Diagnosis of Event-Driven Systems based on Probabilistic Inference

Shinkichi Inagaki and Tatsuya Suzuki
Nagoya University
Japan

1. Introduction

Event-driven controlled systems based on the Programmable Logic Controller (PLC) are widely used in many industrial processes. The number of such a control system is said to occupy more than eighty percent of the entire existing control systems. Nowadays, the demands for production facilities are shifting from the high speed and highly efficiency to the safety and high reliability. In order to meet these requirements, several strategies for fault diagnosis of systems and the design of recovery procedure have been proposed.

In the case of considering the PLC-based control systems, since they have discrete and event-driven characteristics inherently, system models based on discrete-event-system description give more efficient diagnostic algorithm than those based on continuous-time systems (for surveys cf. (A. Darwiche & G. Provan (1996); D. N. Pandalai& L. E. Holloway (2000); M. Sampath et al. (1995); S.H.Zad et al. (1999))). This aspect will be more emphasized when the number of components would be large. Based on these considerations, Lunze proposed a centralized fault diagnosis framework based on the system model with Timed Markov Model (TMM) (J.Lunze (2000)). This method especially becomes useful when numerous number of input and output data are collected through daily operation since the TMM is based on a stochastic expression of time interval between successive events. This approach also has some robustness against unevenness underlying in the ordinary production facilities. However, this kind of centralized diagnosis strategies will cause an explosion of the computational burden when they are applied to the large scale systems. In this case, the decentralized approach is highly recommended wherein the diagnosis is performed by each diagnose together with the communication with other diagnosers (O.Contant (2006); S.Debouk (2000); R.Su et al. (2002)). These approaches, however, were based on the deterministic model.

Based on these backgrounds, the authors (S.Inagaki et al. (2007)) proposed a decentralized stochastic fault diagnosis strategy based on a combination of TMM and Bayesian Network (BN). The BN represents the causal relationship between the fault and observation in subsystems. Since the decentralized diagnosis architecture distributes the computational burden for the diagnosis to the subsystems, a large scale diagnosis problems in real-world application can be solved. In the decentralized approach, the computational burden and the diagnosis performance strongly depend on the complexity of the graph structure of BN.

This chapter also addresses a design method of the graph structure of the BN in decentralized stochastic fault diagnosis of (S.Inagaki et al. (2007)) based on the control logic implemented on the system. For example, an actuator speed reduction affects on the (timed) event sequences observed by the sensors allocated in the subsystems. The effects of this type of fault on other subsystems depend on the control logic wherein the observed event signal is used as a firing condition of the actuators in other subsystems. Thus, the coupling in the control logic over subsystems must be considered in the design of the graph structure of BN. In order to formally realize this idea, the Sensor Actuator Dependency (SAD) graph and the Dependency Tree (DT) are constructed from the control logic in our strategy. The resulting DT represents the hierarchy of the causal relationship between the components in the system. Therefore, by specifying the level of hierarchy appropriately, the graph structure of BN with different level of complexities can be designed.

The remaining part of this chapter is organized as follows: In section 2, we define the problem statement of decentralized fault diagnosis. In section 3, we overview the entire strategy of the fault diagnosis based on BN with a simple example. In section 4, local diagnosis based on TMM is introduced and, in addition, the calculation results of the local diagnosers are combined based on BN. Section 5 shows the procedure of the proposed decentralized diagnosis. In section 6, estimation strategy of probability distribution functions (PDF) which is used in the local diagnosis is introduced based on maximum entropy principle (M.Saito et al (2006)). In section 7, the usefulness of the stochastic decentralized fault diagnosis is verified through some experimental results of an automatic transfer line which is widely used in the industrial world. Section 8 proposes a design method of the graph structure of BN, and, in section 9, the decentralized fault diagnosis is applied to the automatic transfer line, while the system scale is larger than that in section 7, with trying some BN structures which are constructed based on the proposed design method. Section 10 concludes this chapter.

2. Problem statement

First, we assume that the controlled system can be divided into n subsystems in consideration of the architecture of the hardware and/or software. Furthermore, the output (event) sequence, which corresponds to the series of the ON/OFF of sensors and actuators, can be observed in each subsystem. Then, the event sequence for the k -th subsystem $\mathbf{E}_t^k(t_h)$ is defined as follows:

$$\mathbf{E}_t^k(t_h) = (e_0^k, t_0^k; e_1^k, t_1^k; \dots; e_H^k, t_H^k), t_H^k \leq t_h, \quad (1)$$

where e_H^k is the H -th event and t_H^k is the occurrence time of the H -th event in the k -th subsystem. In addition, the κ -th fault in the k -th subsystem is represented by r_{κ}^k , and a combination of faults for all subsystems is defined as “ \mathbf{r} -combination of faults for the entire system.” The set of \mathbf{r} -combination of faults for the entire system \mathcal{R} is defined below:

$$\mathcal{R} = \left\{ \mathbf{r} = (r_{\alpha}^1, r_{\beta}^2, \dots, r_{\nu}^n) \mid \alpha, \beta, \dots, \nu \in \{0, 1, 2, \dots\} \right\}. \quad (2)$$

This paper deals with the following diagnosis problem:

Given : output sequence $E_t^1(t_h), \dots, E_t^n(t_h)$
 Find : fault $r \in \mathcal{R}$

3. Global diagnosis based on Bayesian network

Bayesian Network (BN) is a probabilistic inference network which expresses qualitative causal relations between some random variables by a graph structure together with the conditional probability assigned to each arc (E.Castillo et al. (1997)).

In this section, the proposed global diagnosis method is explained. First, two types of random variables are defined. The first one is R^k which takes r_κ^k ($\kappa \in \{0, 1, \dots, K\}$) as a realization. The second one is the E^k which takes the observed event sequence as a realization. In the BN, the causal relationship between these random variables are defined using a graph structure wherein each node corresponds to each random variable. For the purpose of the fault diagnosis, we restrict the structure of the BN in the bipartite graph. One subset consists of the set of R^k s, and the other subset consists of the set of E^k s (Fig.1). We also assume that there are no causal relationship between nodes in the same subset. The development of an appropriate graph structure must be made by considering the physical and logical interactions between subsystems. The fault diagnosis can be realized by calculating the occurrence probability of each fault conditioned by the observed event sequence.

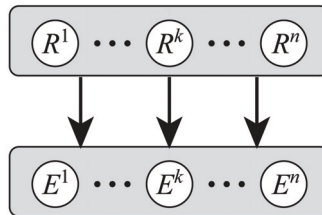


Fig. 1. Bipartite Bayesian Network for fault diagnosis

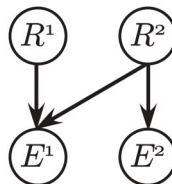


Fig. 2. Example of Bayesian Network

Figure 2 shows the example of the BN for fault diagnosis. The occurrence probability of the fault in the subsystem 1 can be systematically calculated as follows: First, the joint probability distribution (JPD) is uniquely decided based on the graph structure.

$$P(R^1, R^2, E^1, E^2) = P(R^1)P(E^1|R^1, R^2)P(R^2)P(E^2|R^2). \tag{3}$$

Then, the occurrence probability of the fault in the subsystem 1 is calculated by marginalizing the JPD. For example, the fault occurrence probability of the fault r_α^1 in the subsystem 1 is calculated as follows:

$$\begin{aligned}
& P(R^1 = r_\alpha^1 | E^1 = \mathbf{E}_t^1(t_h), E^2 = \mathbf{E}_t^2(t_h)) \\
&= \frac{1}{Z} \left\{ P(R^1 = r_\alpha^1) \sum_{R^2} P(E^1 = \mathbf{E}_t^1(t_h) | R^1 = r_\alpha^1, R^2) \times P(R^2) P(E^2 = \mathbf{E}_t^2(t_h) | R^2) \right\} \quad (4)
\end{aligned}$$

where Z is normalized term and is represented as (5).

$$Z = \sum_{R^1} P(R^1) \sum_{R^2} P(E^1 = \mathbf{E}_t^1(t_h) | R^1, R^2) \times P(R^2) P(E^2 = \mathbf{E}_t^2(t_h) | R^2). \quad (5)$$

In (4), the term $P(E^1 = \mathbf{E}_t^1(t_h) | R^1 = r_\alpha^1, R^2)$ represents the conditional probabilities assigned to the corresponding arc. This conditional probability can be calculated using the local diagnosis results and the Bayesian estimation (see section 4.3 for detail). Also, the prior probabilities (for example $P(R^1 = r_\alpha^1)$ in (4)) are supposed to be given in advance. See section 7.4 for another example.

4. Local diagnosis based on TMM

4.1 Timed Markov model

For the local diagnosis, the relationship between two successive events observed in the corresponding subsystem are represented by means of Timed Markov Model (TMM). The TMM is one of the Markov model wherein the state transition probabilities depend on time. In other words, state transition probabilities vary according to the time interval between two successive events. In the following, representation of the event driven system based on the TMM is briefly described (J.Lunze (2000)).

First of all, the set of fault random variables which are connected to the random variable E^k is defined and denoted by $R^{k_1}, R^{k_2}, \dots, R^{k_m}$. Then, a combination of these realizations is defined as “ \mathbf{r}^k - combination of faults for the k -th subsystem.” Furthermore, the set of these is denoted by $\mathcal{R}^k = \{\mathbf{r}^k = (r_{\kappa_1}^{k_1}, r_{\kappa_2}^{k_2}, \dots, r_{\kappa_m}^{k_m}) | \kappa_1, \kappa_2, \dots, \kappa_m \in \{0, 1, \dots\}\}$. Roughly speaking, \mathbf{r}^k consists of the realization of the faults which affect on the measurement of the k -th subsystem E^k . For example, in Fig.2, $\mathbf{r}^1 = (r_{\kappa_1}^1, r_{\kappa_2}^2)$, and $\mathbf{r}^2 = (r_{\kappa_2}^2)$. Based on definition of the \mathbf{r}^k , the following two functions are defined to specify the stochastic characteristics in the TMM.

Definition 1 A probability density function (PDF) $f_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, \tau^k)$:

$f_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, \tau^k)$ represents a probability density function for the time interval τ^k under the situation that the fault \mathbf{r}^k exists. Note that τ^k is a time interval between two successive events e_{H+1}^k and e_H^k in the k -th subsystem.

Definition 2 A probability distribution function $F_{e_H^k}^k(\mathbf{r}^k, \tau^k)$:

$F_{e_H^k}^k(\mathbf{r}^k, \tau^k)$ represents a probability distribution function that the event e_{H+1}^k does not occur within τ^k after event e_H^k has occurred under the situation that the fault \mathbf{r}^k exists. $F_{e_H^k}^k(\mathbf{r}^k, \tau^k)$ is represented by integrating $f_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, \tau^k)$.

$$F_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, \tau^k) = \int_0^{\tau^k} f_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, t) dt, \quad (6)$$

$$F_{e_H^k}^k(\mathbf{r}^k, \tau^k) = 1 - \sum_{e_{H+1}^k \in \mathcal{E}^k} F_{e_{H+1}^k, e_H^k}^k(\mathbf{r}^k, \tau^k) \quad (7)$$

where some symbols are defined as follows:

e_H^k : H -th event in the k -th subsystem

t_H^k : Occurrence time of event e_H^k

t_h : Sampling time index

τ^k : Waiting time from the occurrence of the latest event in the k -th subsystem ($\tau^k = t_h - t_H^k$)

\mathcal{E}^k : Set of events that occur in the k -th subsystem

Then, relationship between two successive events observed in the subsystem can be described by specifying the probability distribution functions. This function plays an essential role in the TMM based modeling and diagnosis. Section 6 shows an effective estimation method of the probability distribution functions.

4.2 Local diagnosis method

The goal of the local diagnosis is to find the following fault occurrence probability based on the observation only of the k -th subsystem:

$$P(R^{k_1} = r_{\kappa_1}^{k_1}, \dots, R^{k_m} = r_{\kappa_m}^{k_m} | E^k = \mathbf{E}_t^k(t_h)) \equiv p_M^k(\mathbf{r}^k, t_h). \quad (8)$$

Equation (8) represents an occurrence probability of the \mathbf{r}^k conditioned by the observation in the k -th subsystem $\mathbf{E}_t^k(t_h)$. For the calculation of (8), the recursive algorithm has been developed in (J.Lunze (2000)). First, the following two cases must be distinguished:

Case(a): There is no event at time t_h

Case(b): The $(H + 1)$ -th event e_{H+1}^k occurs at time t_h

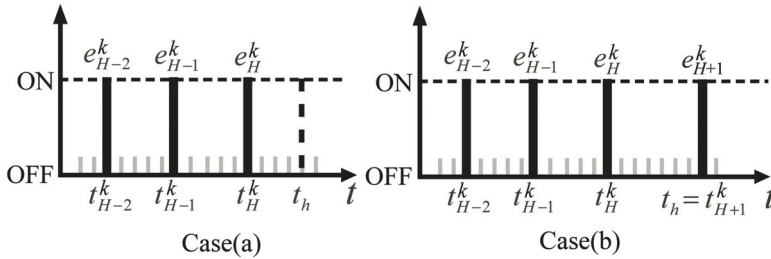


Fig. 3. Time and events in the cases (a) and (b)

Fig. 3 shows relations between time and events in the cases (a) and (b). The diagnosis begins with no information on the existence of the fault, i.e. the initial probabilities are given by

$$p_M^k(\mathbf{r}^k, 0) = \frac{1}{n_{\mathcal{R}^k}} \quad (9)$$

where $n_{\mathcal{R}^k}$ denotes the number of realizations in \mathcal{R}^k . Next, an auxiliary function $p_a^k(\mathbf{r}^k, t_h)$ is calculated as follows:

Case(a) : No event is observed at time t_h

$$p_a^k(\mathbf{r}^k, t_h) = F_{e_H^k}^k(\mathbf{r}^k, t_h - t_H^k) p_M^k(\mathbf{r}^k, t_H^k). \quad (10)$$

Case(b) : The $(H + 1)$ -th event e_{H+1}^k occurs at time t_h

$$p_a^k(\mathbf{r}^k, t_h) = f_{e_{H+1}^k e_H^k}^k(\mathbf{r}^k, t_h - t_H^k) p_M^k(\mathbf{r}^k, t_H^k). \quad (11)$$

The fault occurrence probability given by (8) is updated by

$$p_M^k(\mathbf{r}^k, t_h) = \frac{p_a^k(\mathbf{r}^k, t_h)}{\sum_{\mathbf{r}^k \in \mathcal{R}^k} p_a^k(\mathbf{r}^k, t_h)}. \quad (12)$$

4.3 Calculation of conditional probability in the BN

In the global diagnosis, the calculation of the conditional probability was the key computation (see (4) as an example). The conditional probabilities assigned to each arc (appearing in the marginalized JPD) in the BN can be calculated using (8) and Bayes theorem as follows:

$$P(E^k = \mathbf{E}_i^k(t_h) | R^{k_1} = r_{k_1}^{k_1}, \dots, R^{k_m} = r_{k_m}^{k_m}) = \frac{P(R^{k_1} = r_{k_1}^{k_1}, \dots, R^{k_m} = r_{k_m}^{k_m} | E^k = \mathbf{E}_i^k(t_h)) P(E^k = \mathbf{E}_i^k(t_h))}{P(R^{k_1} = r_{k_1}^{k_1}, \dots, R^{k_m} = r_{k_m}^{k_m})} \quad (13)$$

where the prior probability $P(R^{k_1} = r_{k_1}^{k_1}, \dots, R^{k_m} = r_{k_m}^{k_m})$ is given in advance. Note that the probability $P(E^k = \mathbf{E}_i^k(t_h))$ is not required to be calculated in advance because it is canceled out in (4). This equation implies that the global diagnosis can be executed by integrating results of the local diagnosis.

5. Diagnosis procedure

The procedure of the proposed decentralized diagnosis is depicted in Fig.4. First of all, observe the event sequence in each subsystem. Second, perform the local diagnosis in each subsystem based on the observed event sequence and calculate the conditional probabilities in the BN using (13). Then, calculate the fault occurrence probabilities by means of the BN (global diagnosis). Finally, select the greatest probability among all fault candidates in each subsystem. The diagnosis result for the k -th subsystem is the fault r_i^k that satisfies the following equation in the case that the fault candidates for the k -th subsystem are $\{r_0^k, \dots, r_K^k\}$.

Diagnosis Result for the k -th subsystem

$$= \arg \max_{r_i^k} \left(P \left(R^k = r_0^k | E^1 = \mathbf{E}_i^1(t_h), \dots, E^n = \mathbf{E}_i^n(t_h) \right), \dots, P \left(R^k = r_K^k | E^1 = \mathbf{E}_i^1(t_h), \dots, E^n = \mathbf{E}_i^n(t_h) \right) \right). \quad (14)$$

6. Estimation of probability density function by maximum entropy principal

As described in the preceding sections, it is required to estimate all probability distribution functions (PDF) $f_{e_{H+1} e_H}(\mathbf{r}, \tau)$ in advance for modeling the system based on TMM, where the

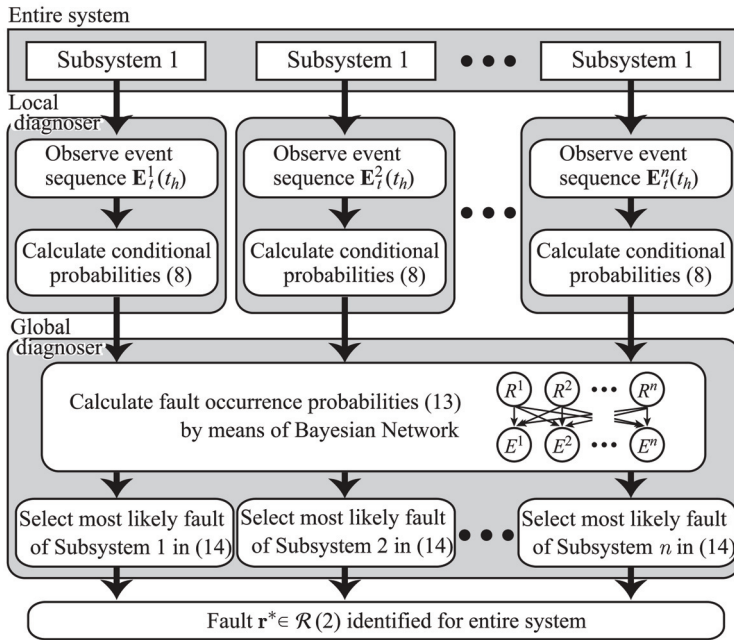


Fig. 4. Procedure of the decentralized fault diagnosis

superscript k representing subsystem k is omitted for simplicity in this section. One of the most straightforward way to do it is to collect numerous number of output sequences, and generate the histogram of the time interval of all two successive events for various situations such as normal or some kind of faulty. In the real application, it is not necessary to collect data for all situations in advance. When some new fault occur, then the new observed data for the new fault can be simply added to the old database as for the PDF. Thus, the PDF can be updated according to the occurrence of the new fault.

Although, the PDF $f_{e_{H+1}e_H}(\mathbf{r}, \tau)$ can be estimated by collecting the observed output sequence, when we consider to use it as the system model, we often face the zero frequency problem which leads to incorrect result in the system diagnosis based on TMM. In order to overcome this problem, the maximum entropy principle (M.Saito et al (2006)) is introduced in this section. It enables us to find the PDF $f_{e_{H+1}e_H}(\mathbf{r}, \tau)$, which maximizes the entropy with keeping the stochastic characteristics of the collected observed data (i.e. the histogram). The remaining part of this section is devoted to describe the estimation procedure for PDF by means of the maximum entropy principle.

First of all, a histogram is created based on observed data. Then, a range of τ , $[\mu - 3\sigma, \mu + 3\sigma]$ is quantized into n equal intervals under the assumption that all unknown data exists in $[\mu - 3\sigma, \mu + 3\sigma]$ where μ and σ are mean value of the observed data and standard deviation, respectively.

Second, let $\{\tau_1, \tau_2, \dots, \tau_n\}$ be the center of each interval, and let $\{f_{e_{H+1}e_H}(\mathbf{r}, \tau_1), f_{e_{H+1}e_H}(\mathbf{r}, \tau_2), \dots, f_{e_{H+1}e_H}(\mathbf{r}, \tau_n)\}$ be the probabilities corresponding to the points $\{\tau_1, \tau_2, \dots, \tau_n\}$. The example of this quantization is illustrated in Fig.5.

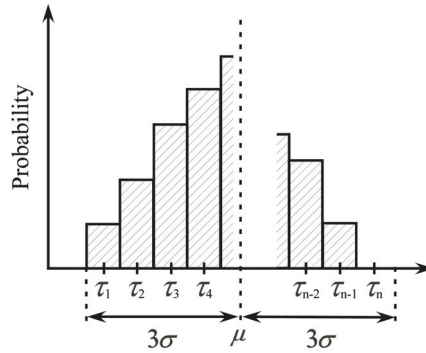


Fig. 5. Time interval of event transition

Finally, we solve the following entropy maximization problem:

Find $f_{e_{H+1}e_H}(\mathbf{r}, \tau_i)$ which maximizes

$$S = - \sum_{i=1}^n f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) \log f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) \quad (15)$$

subject to

$$\begin{aligned} \sum_{i=1}^n f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) &= 1, \\ \sum_{i=1}^n (\tau_i)^j \cdot f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) &= a_j \quad (j = 1, 2, \dots, J), \\ f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) &\geq 0, \end{aligned} \quad (16)$$

where $a_j (= E[(\tau)^j])$ is the j -th moment obtained from the observed data. This problem can be solved by applying the Lagrange multiplier method, and the solution has a form given by

$$f_{e_{H+1}e_H}(\mathbf{r}, \tau_i) = \exp[-\lambda_0 - \lambda_1 \tau - \lambda_2 (\tau)^2 - \dots - \lambda_m (\tau)^J]. \quad (17)$$

where λ_0 is given by (18) and $\lambda_1, \dots, \lambda_m$ are the Lagrange multipliers corresponding to the m constraints.

$$\lambda_0 = \log \left(\sum_{i=1}^n \exp \left(- \sum_{j=1}^m \lambda_j (\tau_i)^j \right) \right) \quad (18)$$

The estimated PDFs are applied to the interval $[\mu - 3\sigma, \mu + 3\sigma]$. For the outside of the range $[\mu - 3\sigma, \mu + 3\sigma]$, probabilities are set to be zero and ε in normal and faulty situations, respectively.

Figs.6 and 7 show PDF examples constructed by observed data in a transfer machine (see section 7 for details). Then, several moment constraints given by (16) were specified by using the histogram. In these examples, 1st and 2nd moments were considered. The problem of entropy maximization (15) was solved by using the Lagrange multiplier method. Estimated PDF are given by (19) and (20), respectively, where ε is 0.01. Thick solid line in Figs.6 and 7 represent the estimated PDF.

$$f_{e_{65}e_{21}}(r_0, \tau) = \begin{cases} \exp(-2.184 + 2.237\tau - 1.348\tau^2) & \text{for } 0 \leq \tau \leq 2.51 \\ 0 & \text{for } 2.51 < \tau \end{cases} \quad (19)$$

$$f_{e_{54}e_{10}}(r_2, \tau) = \begin{cases} \exp(-22.67 + 16.73\tau - 3.250\tau^2) & \text{for } 1.41 \leq \tau \leq 3.72 \\ \varepsilon & \text{else} \end{cases} \quad (20)$$

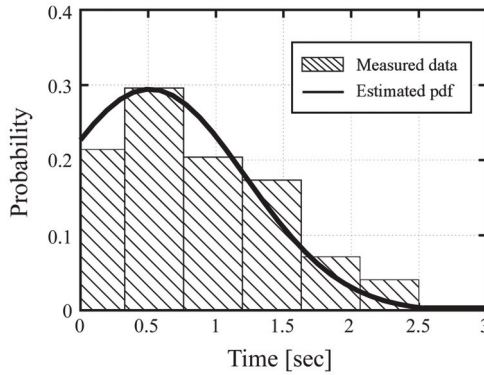


Fig. 6. Histogram and PDF ($f_{e_{65}e_{21}}(r_0, \tau)$)

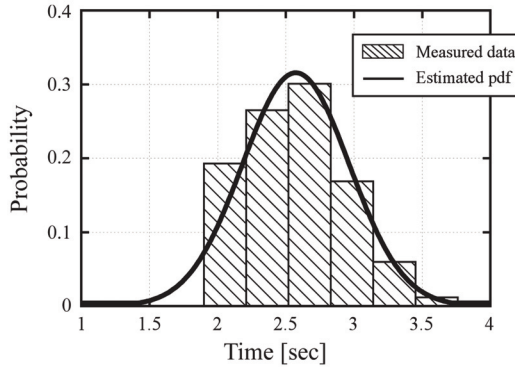


Fig. 7. Histogram and PDF ($f_{e_{54}e_{10}}(r_2, \tau)$)

7. Application to automatic transfer line

In this section, the proposed diagnosis procedure is applied to the automatic transfer line depicted in Fig.8. This type of machine is widely used in industrial world.

7.1 Automatic transfer line

Fig.9 shows the diagram of the developed prototype transfer line shown in Fig.8. This system transfers works to the unload station by means of two belt-conveyors (L1, L2: their length are 50cm) and two cranes (C1, C2). Sensors (S1-S6) are installed at the beginning, end and center of the conveyors and the sensor S7 is installed at the unload station. The events are observed when the work crosses the sensors, and are depicted also in Fig.9 superimposing on the automatic transfer line.

The transfer line system is decomposed into the four subsystems (Lane1, Crane1, Lane2, Crane2) as shown in Fig.10. The set of events observed in each subsystem is specified in Table 1.

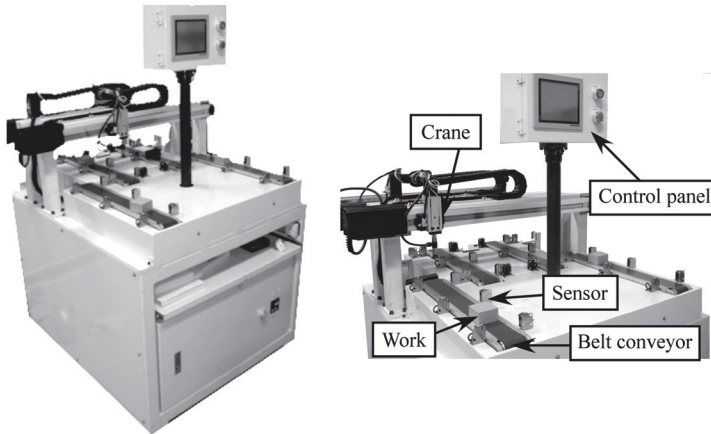


Fig. 8. Prototype of automatic transfer line

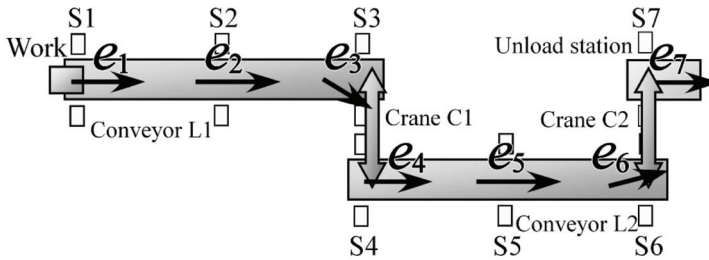


Fig. 9. Diagram of the transfer line and definition of events

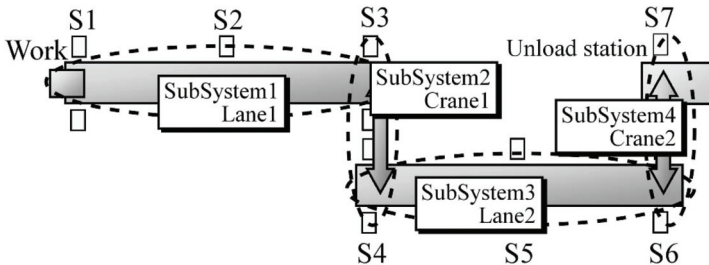


Fig. 10. Definition of subsystems

7.2 Candidates of fault

We consider the candidates of fault in each subsystem specified in Table 2. Note that it is unlikely that these faults are diagnosed using deterministic approach.

For the Lane1 and Lane2, the “normal” implies the case that the speed of the belt-conveyor is between 7.8cm/sec and 8.6cm/sec, and the “Speed of the belt-conveyor is reduced”

implies the case that the speed of the belt-conveyor goes down between 7.0cm/sec and 7.8cm/sec. Faults r_1^1 and r_1^3 may come from a fatigue of the actuator. The "Sensor does not respond with probability of 50%" may occur by means of a defective wiring and so on. This corresponds to the early stage of the fatal fault wherein the sensor does not respond at all. Thus, $3 \times 1 \times 3 \times 2 = 18$ fault cases are investigated for the entire system including cases that some faults occur simultaneously in some subsystems.

7.3 Experimental conditions

Experimental conditions are specified as follows:

- Works are provided to the line with almost constant intervals (about 5 sec).
- Works do not exist in the system at time $t_n = 0$.
- The experiment is finished if ten works are transferred to the unload station.
- A sampling time for observation of events is 0.1 sec.

Under these experimental conditions, the event sequences are collected. The probability density functions (PDFs) for every combination of two successive events in each subsystem are estimated before fault diagnoses. The PDFs are estimated through eighty trials per each fault case in advance.

7.4 Graph structure

As mentioned in section 3, two types of random variables are defined and specified as nodes in the BN. The first one is R^k which takes r_κ^k ($\kappa \in \{0, 1, \dots, K\}$) as a realization. The second one is the E^k which takes the observed event sequence as a realization. In this application, a graph structure depicted in Fig.11 is adopted under the consideration that the faults occurred in the k -th subsystem influence on the event sequences observed in the $(k - 1)$ -th and the k -th subsystem. Generally speaking, the graph structure should be designed from viewpoints of the computational burden for the diagnosis and the hardware / software interactions between subsystems. Development of the formal procedure for the generation of the graph structure is now under investigation.

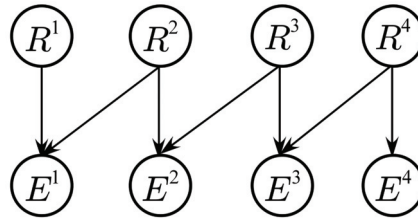


Fig. 11. Graph structure of the BN for the transfer line

The JPD is calculated based on Fig.11 as follows:

$$\begin{aligned} & P(R^1, R^2, R^3, R^4, E^1, E^2, E^3, E^4) \\ & = P(R^1)P(R^2)P(R^3)P(R^4)P(E^1|R^1, R^2)P(E^2|R^2, R^3)P(E^3|R^3, R^4)P(E^4|R^4). \end{aligned} \quad (21)$$

Then, the probabilistic inference based on the BN becomes possible by marginalizing the JPD. For example, the occurrence probability of the fault r_0^1 in the subsystem 1 is calculated as follows:

$$P(R^1 = r_0^1 | E^1 = \mathbf{E}_i^1(t_h), \dots, E^4 = \mathbf{E}_i^4(t_h)) = \frac{1}{Z_1} \left\{ P(R^1 = r_0^1) \sum_{R^2} \sum_{R^3} \sum_{R^4} P(R^2) P(R^3) P(R^4) \right. \\ \left. \times P(E^1 = \mathbf{E}_i^1(t_h) | R^1 = r_0^1, R^2) P(E^2 = \mathbf{E}_i^2(t_h) | R^2, R^3) P(E^3 = \mathbf{E}_i^3(t_h) | R^3, R^4) P(E^4 = \mathbf{E}_i^4(t_h) | R^4) \right\} \quad (22)$$

where Z_1 is normalized term, and is represented by

$$Z_1 = \sum_{R^1} \left\{ P(R^1) \sum_{R^2} \sum_{R^3} \sum_{R^4} P(R^2) P(R^3) P(R^4) P(E^1 = \mathbf{E}_i^1(t_h) | R^1, R^2) P(E^2 = \mathbf{E}_i^2(t_h) | R^2, R^3) \right. \\ \left. \times P(E^3 = \mathbf{E}_i^3(t_h) | R^3, R^4) P(E^4 = \mathbf{E}_i^4(t_h) | R^4) \right\}. \quad (23)$$

SubSystem	Set of events	SubSystem	Set of events
Lane1	$\mathcal{E}^1 = \{e_1, e_2, e_3\}$	Crane1	$\mathcal{E}^2 = \{e_3, e_4\}$
Lane2	$\mathcal{E}^3 = \{e_4, e_5, e_6\}$	Crane2	$\mathcal{E}^4 = \{e_6, e_7\}$

Table 1. Set of events in each subsystem

Symbol	Detail of fault
r_0^1	Lane1 is normal.
r_1^1	Speed of the belt-conveyor L1 is reduced.
r_2^1	Sensor S2 does not respond with probability of 50%.
r_0^2	Crane1 is normal.
r_0^3	Lane2 is normal.
r_1^3	Speed of the belt-conveyor L2 is reduced.
r_2^3	Sensor S5 does not respond with probability of 50%.
r_0^4	Crane2 is normal.
r_1^4	Works fall from crane C2 with probability of 50%.

Table 2. Candidates of fault

Method	Success	Wrong Diagnosis	Undetection
Decentralized Method	81.0%	12.7%	6.3%
Centralized Method	92.9%	2.4%	4.7%

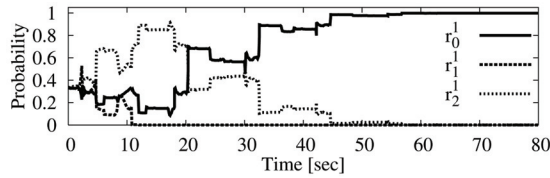
Table 3. Comparison between decentralized method (proposed) and centralized method (conventional)

7.5 Results of fault diagnosis

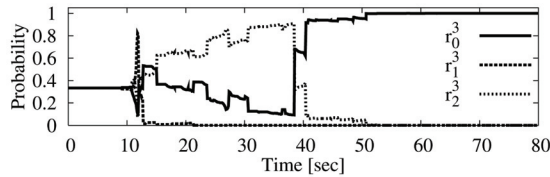
7.5.1 Faultless case

Fig.12 shows the profiles of the fault occurrence probability in each subsystem wherein no fault has occurred in the entire controlled system. The result in the subsystem 2 (Crane 1) is

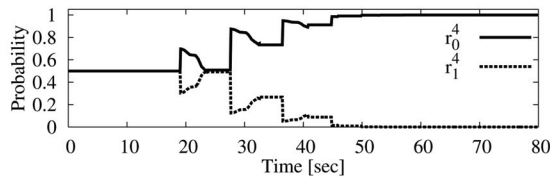
eliminated because no fault has considered in the subsystem 2. In Figs.12(a) to 12(c), the probability of the “normal (r_0^k)” becomes almost 1 before 45 sec in all subsystems, and this result lasts until the experiment is completed. This implies that the result of the diagnosis for all subsystems are “normal (r_0^k)”, and agrees with the actual situation of the system.



(a) Diagnosis result for Lane1



(b) Diagnosis result for Lane2

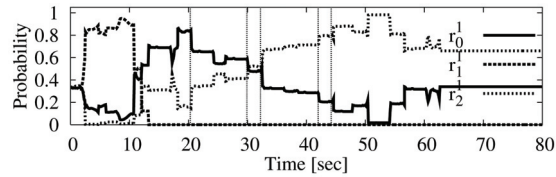


(c) Diagnosis result for Crane2

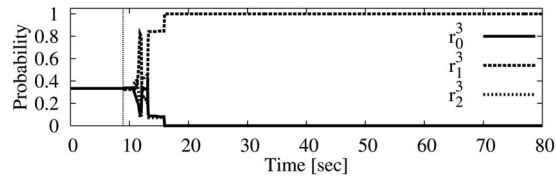
Fig. 12. Diagnosis result in the faultless case: $\mathbf{r} = (r_0^1, r_0^2, r_0^3, r_0^4)$

7.5.2 Multiple faulty case

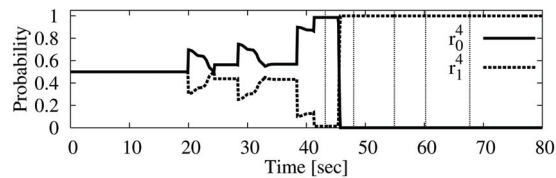
Fig.13 shows the profiles of the fault occurrence probability in each subsystem wherein the faults r_2^1 , r_3^3 and r_1^4 have occurred at a certain time (no fault has occurred in the subsystem 2). In Figs.13(a), 13(b) and 13(c), the vertical lines represent the time instants when the faults r_2^1 , r_3^3 , and r_1^4 occurred, respectively. In the subsystem 1 (Fig.13(a)), the probability of the fault r_2^1 goes up every time when the fault occurs, and shows the greatest probability when the experiment is completed. As the result, the fault r_2^1 can be uniquely identified in the subsystem 1. Furthermore, in the subsystem 3 (Fig.13(b)) and subsystem 4 (Fig.13(c)), the faults r_3^3 and r_1^4 can be identified successfully a few seconds after each fault has occurred. These results show that the diagnosis results completely agree with the actual faulty situation.



(a) Diagnosis result for Lane1



(b) Diagnosis result for Lane2



(c) Diagnosis result for Crane2

Fig. 13. Diagnosis result in the faultless case: $\mathbf{r} = (r_2^1, r_0^2, r_1^3, r_1^4)$

7.5.3 Comparison with centralized method

We have performed the experiments seven times for each fault, i.e., the total number of the trials is $7 \times 18 = 126$. The statistics of the diagnosis results are listed in Table 3 together with the statistics of the centralized approach (M.Saito et al (2006)) (i.e. the system is not decomposed). In Table 3, the "Success Rate" means the rate that the all diagnosis results coincide with the actual fault situation, the "Wrong Diagnosis Rate" means the rate that at least one of the subsystems had wrong diagnosis result, and the "Undetection Rate" means the rate that the diagnosis result was "normal" in spite of existence of the fault. The success rate of proposed decentralized diagnosis is 81%. This is reduced by 13% compared with the conventional centralized method. This reason is considered that the direct relationships (arcs) between R^k and R^ℓ ($k \neq \ell$), E^k and E^ℓ ($k \neq \ell$) are ignored. However, using the proposed decentralized strategy can distribute the computational burden for the diagnosis to the subsystems with sacrificing the small degradation of the success rate. The appropriate selection of the graph structure in the BN will lead to the increase of the success rate.

8. Design of graph structure

In this section, the graph structure of the BN is designed based on the control law applied to the controlled system. The design procedure is explained step by step with an example. The controlled system is defined by three tuples as follows:

$$G = \{S, A, C\} \tag{24}$$

where S is the set of sensors, A is the set of actuators, and C is the set of control laws. The system is divided into subsystems:

$$G^k = \{S^k, A^k, C^k\}, G = G^1 \cup G^2 \cup \dots \cup G^n \tag{25}$$

where A^k and S^k are the set of actuators and sensors included in the k -th subsystem, respectively. In addition, C^k is the set of control laws relevant to A^k . Figure 14 shows the diagram of the developed prototype transfer line. This system transfers works to the unload station by means of six actuators; four lanes (Lane1-Lane4; their length are 50 cm) and two cranes (Crane1, Crane2). Sensors (S1-S12) are installed at the beginning, end and center of the lanes, and the sensor S13 is installed at the unload station. The events depicted in Fig.14 are observed when the work crosses the sensors. The transfer line system is decomposed into six subsystems as shown in Fig.14. The set of events observed in each subsystem is specified in Table 4.

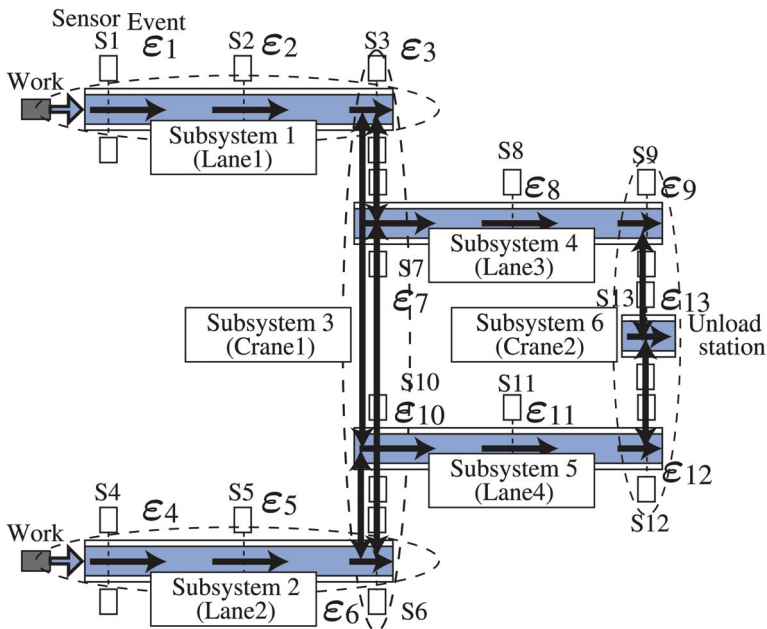


Fig. 14. Diagram of transfer line and definition of events

Lane1	$\mathcal{E}^1 = \{\epsilon_1, \epsilon_2, \epsilon_3\}$	Lane2	$\mathcal{E}^2 = \{\epsilon_4, \epsilon_5, \epsilon_6\}$
Crane1	$\mathcal{E}^3 = \{\epsilon_3, \epsilon_6, \epsilon_7, \epsilon_{10}\}$	Lane3	$\mathcal{E}^4 = \{\epsilon_7, \epsilon_8, \epsilon_9\}$
Lane4	$\mathcal{E}^5 = \{\epsilon_{10}, \epsilon_{11}, \epsilon_{12}\}$	Crane2	$\mathcal{E}^6 = \{\epsilon_9, \epsilon_{12}, \epsilon_{13}\}$

Table 4. Set of events in each subsystem

The control laws applied to the system are summarized as follows:

- Each lane is interlocked by its terminal sensor, i.e., stops when the terminal sensors (S3, S6, S9 and S12) are fired.
- The lane continues to behave in the absence of the interlock stop.
- Lane3/Lane4 stop when Crane1 is moving down at the position S7/S10.
- Each crane starts to move and to transfer a work when a work reaches at the terminal sensor.
- Crane1 transfers a work from Lane1 or Lane2 to Lane3 or Lane4.
- Crane2 transfers a work from Lane3 or Lane4 to the unload station.
- The crane transfers a work to the nearest lane which is available.

These control laws can be described using the form of a ladder logic ?. For example, Fig.15 shows the ladder logic of the C^1 wherein the operating situation of the Lane1 (L1) is expressed by $L1 = (X \vee L1) \wedge \overline{S3}$. In other case, the logic of the C^4 wherein the operating situation of the Lane3 (L3) is expressed by $L3 = (X \vee L3) \wedge \overline{S9} \wedge \overline{C1}$. This is due to the logic that Lane3/Lane4 stop when Crane1 is moving down at the position S7/S10.

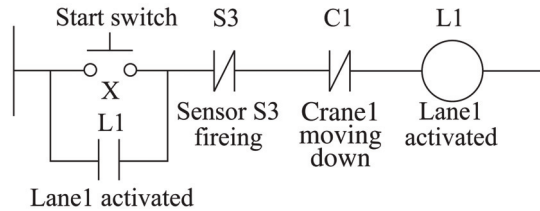


Fig. 15. Ladder logic of control law C^1

Based on this logical relationship between sensors and actuators, the causal relationships between sensors and actuators are extracted and expressed by a sensor actuator dependency (SAD) graph by using the following algorithm:

Algorithm 1: Construction of SAD graph

- Step 1** For all $k = 1, \dots, n$, allocate the set of sensors S^k on the left side and the actuator A^k on the right side.
- Step 2** For all $k = 1, \dots, n$, draw a dashed arrow from A^k to S^h when the controller C^h includes the operating condition of A^k .
- Step 3** For all $k = 1, \dots, n$, draw a solid arrow from S^h to A^k when C^k includes the sensor $s^h \in S^h$ as the starting or halting condition of A^k ,

An example of the SAD graph constructed from the control logic is shown in Fig.16. In the next step, a dependency tree (DT) is produced from the SAD graph by the following algorithm:

Algorithm 2: Construction of DT

Step 1 Set $k = 1$.

Step 2 Set $l = 1$.

Step 3 Allocate S^h underneath A^k , and set the level to be l when S^h is connected to A^k by a dashed arrow in the SAD graph and has not appeared in the level less than l .

Step 4 Allocate A^m underneath S^h of Level l when A^m is connected to S^h by a solid arrow in the SAD graph, and has not appeared in the level less than l .

Step 5 Go to Step 6 if no actuator exists in Step 4, else go to Step 3 with $l = l + 1$.

Step 6 Terminate the algorithm if $k = n$, else go to Step 2 with $k = k + 1$.

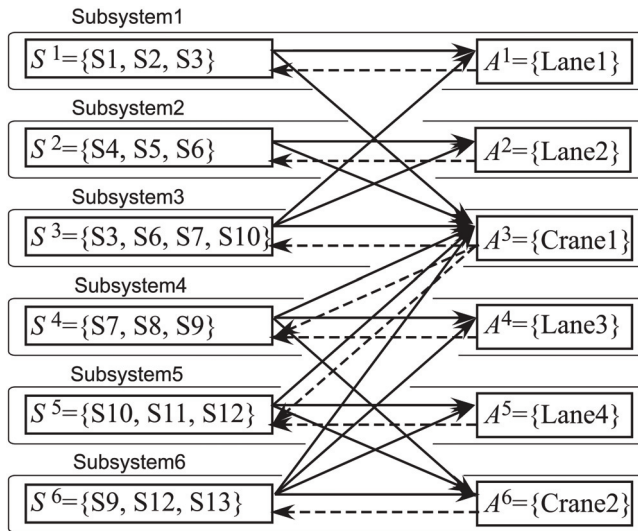


Fig. 16. Sensor actuator dependency (SAD) graph

An example of the DT produced from Fig.16 is shown in Fig.17. In the last step, the structure of BN is designed from the DT by the following algorithm:

Algorithm 3: Design of graph structure of BN

Step 1 For all $k = 1, \dots, n$, allocate the nodes of the random variable R^k and E^k on the upper and lower sides, respectively.

Step 2 For all $k = 1, \dots, n$, draw an arrow from R^k to E^h for all h where S^h is included in the level 1 to L of A^k 's DT.

In this algorithm, the parameter L is a depth of the DT and represents a threshold to take into consider the causal relationship between the subsystems into the graph structure of the BN. Figure 18 is the resultant graph structure when $L = 2$ for the DT in Fig.17. In Fig.18, for example, there exist arcs from R^6 to E^3, E^4, E^5 , and E^6 because S^3, S^4, S^5 , and S^6 are included within Level 2 in Fig.17. Note that although the DT in Fig.17 starts from the actuator, a DT

which starts from the sensor is simply constructed by straightforward modification of Algorithm 2.

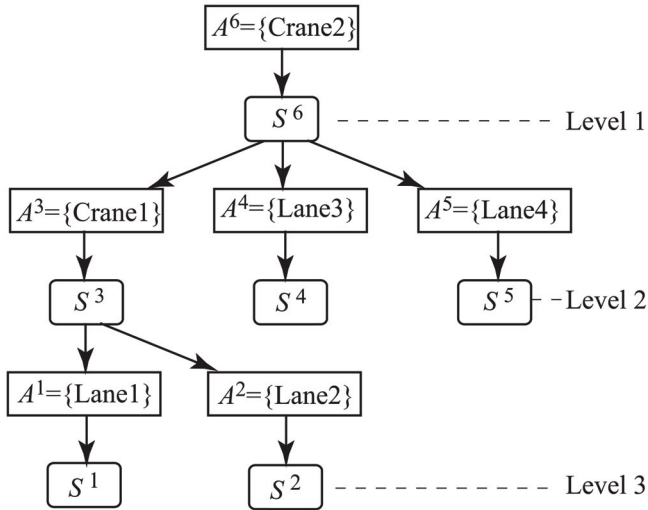


Fig. 17. Dependency tree for Crane2 (Subsystem 6)

9. Experimental verification

In this section, the decentralized diagnosis procedure is applied to the automatic transfer line depicted in Fig.14. The diagnosis procedure is executed by means of three graph structures. Graph structure 1 depicted in Fig.18 is derived in Section 8. Graph structure 2 depicted in Fig.19 considers all causal relationships, i.e., $L = \infty$ in the DT. Graph structure 3 depicted in Fig.20 represents the completely independent diagnosis.

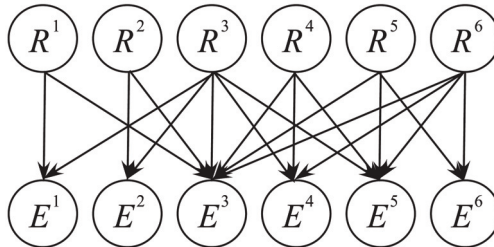


Fig. 18. Graph structure 1

9.1 Candidates of fault

We consider the candidates of fault in each subsystem specified in Table 5. For the lane, the “normal” implies the case that the speed is between 7.8 cm/sec and 8.6 cm/sec, and the “Speed of the lane is reduced” implies the case that the speed goes down between 7.0 cm/sec and 7.8 cm/sec. Faults r_1^2 and r_1^5 may come from a fatigue of the actuator. For the crane, the “Speed of the crane is reduced” implies the case that it takes 0.2 more seconds

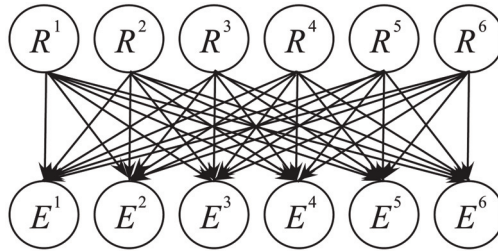


Fig. 19. Graph structure 2

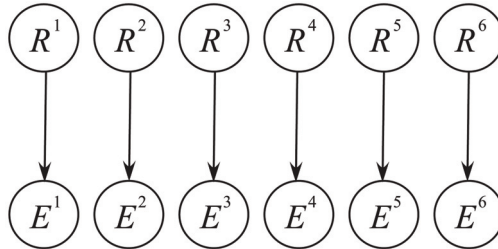


Fig. 20. Graph structure 3

Symbol	Detail of fault
r_0^1	Lane1 is normal
r_0^2	Lane2 is normal
r_1^2	Speed of Lane2 is reduced
r_0^3	Crane1 is normal
r_1^3	Speed of Crane1 is reduced
r_0^4	Lane3 is normal
r_0^5	Lane4 is normal
r_1^5	Speed of Lane4 is reduced
r_0^6	Crane2 is normal
r_1^6	Speed of Crane2 is reduced

Table 5. Candidates of faulty situation

than the “normal” situation to transfer a work to the destination lane. Thus, $1 \times 2 \times 2 \times 1 \times 2 \times 2 = 16$ faulty cases are investigated for the entire system including cases that some faults occur simultaneously among some subsystems.

9.2 Experimental conditions

Experimental conditions are specified as follows:

- Works are provided to the Lane1 and Lane2 alternately with almost constant intervals (about 5 sec).
- Works do not exist in the system at time $t_i = 0$.
- The experiment is finished when twenty works are transferred to the unload station.
- A sampling time for observation of events is 0.1 sec.

All the prior probabilities $P(R^{k_i} = r_{k_i}^{k_i})$ ($i = 1, 2, \dots, m$) in (13) are set to be

$$P(R^{k_i} = r_{k_i}^{k_i}) = \frac{1}{n_{\mathcal{R}^k}}. \tag{26}$$

This means that no statistical information about the faults has not been used for the diagnosis. Under these experimental conditions, the event sequences are collected. The probability density functions (PDFs) for every combination of two successive events in each subsystem are estimated before the fault diagnosis. The PDFs are estimated through fifty trials per each faulty case in advance. The calculation of the diagnosis was performed by personal computers (Pentium 4 2.39 GHz).

9.3 Results of fault diagnosis

We have performed the experiments ten times for each faulty case, i.e., the total number of the trials is $10 \times 16 = 160$. The statistics of the diagnosis results are listed in Table 6. In Table 6, the "Success Rate" means the rate that the all diagnosis results coincide with the actual faulty situation, the "Wrong Diagnosis Rate" means the rate that at least one of the subsystems had wrong diagnosis result, and the "Undetection Rate" means the rate that the diagnosis result was "normal" in spite of existence of the fault.

The success rate of the graph structure 1 and 2 are both increased compared with the structure 3. This is due to the consideration of the causal relationships between subsystems. The structure 2 is better than the structure 1 from viewpoint of the success rate, however, the number of PDFs of the structure 1 is almost half of that of the structure 2. Since the number of the PDFs is related with the computational burden for the real-time inference, the structure 1 can be realized with less computational burden than the structure 2. The computing time shown in Table 6 is the total required time to diagnose the 150.5 [sec] data. These times were obtained from the maximum computing time of each local diagnoser and the computing time of the global diagnoser as shown in Fig.21 (in the case of the structure 1). The computation of the local diagnosers is dominate in the computation of entire

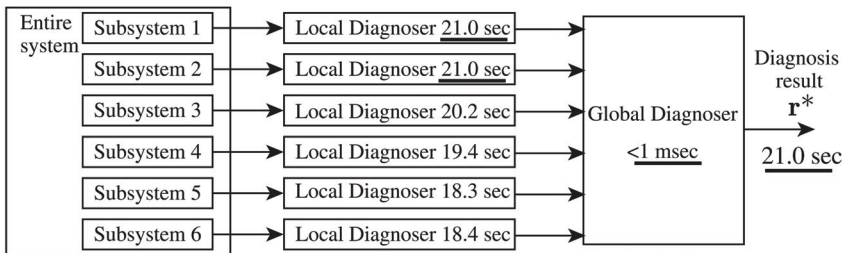


Fig. 21. Computing time for diagnosing 150.5 sec data in graph structure 1

Graph structure	Success	Wrong	Undetected	Number of PDFs	Computing time
1	91.3 %	8.1 %	0.6 %	482	21.0 sec
2	94.4 %	3.8 %	1.9 %	976	51.7 sec
3	86.3 %	7.5 %	6.3 %	104	1.8 sec

Table 6. Comparison of diagnosis results for three graph structures: Computing time for diagnosing 150.5 [sec] data

diagnosis. In addition, the computational burden of the local diagnosers increases by $\prod_{i=1}^{m_k} f_{k_i} \times N(\mathcal{E}_k) C_2$ where $N(\mathcal{E}_k)$ is the number of events in the subsystem k . The level threshold L of Algorithm 3 should be selected from the both viewpoint of the success rate and the computational burden.

10. Conclusions

This paper presented a design method of the graph structure of the Bayesian Network (BN) in the decentralized stochastic fault diagnosis of large-scale event-driven controlled systems. First, in order to estimate the probability density functions of the randomized time intervals, the maximum entropy principle was introduced, which can estimate probability density functions so as to maximize the uniformity with satisfying the constraints caused by observed data.

Second, the controlled plant was decomposed into some subsystems, and the global diagnosis was formulated using the Bayesian Network (BN), which represents the causal relationship between the fault and observation between subsystems.

Third, the local diagnoser was developed using the conventional Timed Markov Model (TMM), and the local diagnosis results were used to specify the conditional probability assigned to each arc in the BN. By exploiting the decentralized diagnosis architecture, the computational burden for the diagnosis can be distributed to the subsystems. As the result, large scale diagnosis problems in the practical situation can be solved.

Forth, the graph structure of the BN is designed based on the control logic applied to the system. In order to realize this, the Sensor Actuator Dependency (SAD) graph and the Dependency Tree (DT) are constructed from the control logic. Since the computational burden and the diagnosis performance mainly depend on the complexity of the graph structure of BN, they are adjusted adequately by specifying the depth of the DT which represents the strength of the causal relationship between components in subsystems.

Finally, the usefulness of the proposed strategy has been verified through some experimental results of an automatic transfer line. Our future work is to verify the decentralized stochastic fault diagnosis strategy in larger scale event-driven controlled systems.

11. References

- A.Darwiche, G.Provan; "Exploiting system structure in model-based diagnosis of discrete-event systems", In *Proc. 7th Intl. Workshop on Principles of Diagnosis.*, pp.95-105, 1996.
- D. N.Pandalai, L.E.Holloway; "Template Languages for Fault Monitoring of Timed Discrete Event Processes", *IEEE Trans. Automa. Contr.*, Vol.45, No.5, pp.868-882, 2000.
- M.Sampath, R.Sengupta, S.Lafortune, K.Sinnamohideen, D.Teneketxis; "Diagnosability of Discrete Event Systems", *IEEE Tras. Automa. Contr.*, Vol.40, No.9, pp.1555-1575, 1995.
- S.H.Zad, R.H.Kwong, W.M.Wonham; "Fault Diagnosis in Timed Discrete-Event Systems", In *Proc. 38th IEEE Conf. Decision Contr.*, pp.1756-1761, 1999.
- J.Lunze; "Diagnosis of Quantized Systems Based on a Timed Discrete-Event Model", *IEEE Trans. Syst. Man. Cybern.*, Vol.30, No.3, pp.322-335, 2000.

- O.Contant, S.Lafortune, D.Teneketzis; "Diagnosability of Discrete Event Systems with Modular Structure", *Discrete Event Dynamic Systems: Theory and Applications.*, Vol.16, No.1, pp.9-37, 2006.
- S.Debouk, S.Lafortune, D.Teneketzis; "Coordinated decentralized protocols for failure diagnosis of discrete-event systems", *Discrete Event Dynamic Systems: Theory and Applications.*, Vol.10, No.1-2, pp.33-86, 2000.
- R.Su, W.M.Wonham, J.Kurien, X.Koutsoukos; "Distributed Diagnosis for Qualitative Systems", In *Proc. 6th International Workshop on Discrete Event Systems.*, pp.169-174, 2002.
- E.Castillo, J.M.Gutiérrez, A.S.Hadi; "Expert Systems and Probabilistic Network Models", Springer, 1997.
- S.Inagaki, T.Suzuki, M.Saito, T.Aoki, "Local/Global Fault Diagnosis of Event-Driven Controlled Systems based on Probabilistic Inference", In *Proc. 46th IEEE Conference on Decision and Control*, pp. 2633-2638, 2007.
- M. Saito, T.Suzuki, S.Inagaki, T.Aoki; "Fault Diagnosis of Event-Driven Control Systems based on Timed Markov Model with Maximum Entropy Estimation", In *Proc. 17th International Symposium on Mathematical Theory of Networks and Systems*, 2006

New Applications Using PLCs in Access Networks

Lamartine V. de Souza, João C. W. A. Costa and Carlos R. L. Francês
Federal University of Pará (UFPA)
Brazil

1. Introduction

Access Networks in telecommunications, such as digital subscriber lines (DSL) and wireless broadband networks (WBN) have become so popular that these systems are now found in almost all regions. The widespread use of these systems has brought about the need for research into new ways of resolving, or at the very least, minimizing the impact of problems that affect the performance of these systems.

In terms of DSL systems, crosstalk is one of the main performance limiting factors, principally when operating at high frequencies, as is the case with VDSL (very-high-bit-rate DSL) networks. Consequently, the required high data rates of VDSL systems may not be achievable if crosstalk levels are excessive.

Across WBN systems, the existence of co-channel interference increases the system's noise levels and also degrades the network's overall performance. It may be impossible therefore, depending on the noise level, to get even minimum system access. It is therefore necessary to plan a way of controlling these noise levels across both access networks.

Programmable logic controllers (PLCs) are the main types of controllers used within the industry. One of their characteristics is the fact that they can operate within aggressive environments (for example, at high temperatures or within high humidity levels) as well as having high operational speeds in comparison with corresponding electro-mechanic control systems; the PLC becoming a highly efficient control device with multiple usage possibilities.

Hence, the use of PLCs across access networks opens up additional fields of application for this type of device, especially due to the fact that up until now, the PLC's widest form of use has been in the industrial sector. Additionally, the robustness, flexibility and speed of the PLC allows it to be used across access networks without any additional need for major configuration changes to already installed equipment, i.e.; the implementation of a PLC into a system does not generate excessive costs or require excessively specialized configurations. PLC application will focus on automated configurations in order to reduce system noise on access networks (DSL and WBN) with the intention of making sure the performance levels of these systems are not degraded in any way and are also able to operate within the expected performance parameters.

In this chapter, we propose alternative PLC applications on two types of broadband networks. Basic concepts about DSL networks and wireless broadband networks are presented in section 2. In section 3 the application of PLC on broadband networks is discussed. Final comments are presented in section 4.

2. Access networks

2.1 DSL networks

DSL access technologies have been developed by the telephone companies to provide high-speed data rates over regular telephone wires. The term DSL covers a number of similar yet competing forms of DSL; including ADSL (asymmetric DSL), SHDSL (single-pair high speed DSL) and VDSL (Starr et al., 1999). These types of DSLs can be summarized as shown in Table 1 (Gonzalez, 2008).

Technology	Name	Ratified	Maximum speed capabilities
ADSL	Asymmetric Digital Subscriber Line, G.dmt	1999	6 Mbps (downstream) 800 kbps (upstream)
ADSL2	G.dmt.bis	2002	8 Mbps (downstream) 1 Mbps (upstream)
ADSL2+	ADSL2plus	2003	24 Mbps (downstream) 1 Mbps (upstream)
ADSL2-RE	Reach Extended	2003	8 Mbps (downstream) 1 Mbps (upstream)
SHDSL	Symmetric High-Bit Rate DSL	2003	5.6 Mbps (downstream/upstream)
VDSL1	Very-high-data-rate DSL 1	2004	55 Mbps (downstream) 15 Mbps (upstream)
VDSL2 -12 MHz long reach	VDSL 2	2005	55 Mbps (downstream) 30 Mbps (upstream)
VDSL2 - 30 MHz short reach	VDSL 2	2005	100 Mbps (downstream/upstream)

Table 1. DSL technology options

Some authors (Ödling et al., 2009) indicate a fourth broadband generation concept with data rates from around 100 Mbps to around 1 Gbps. In this case, broadband systems will operate on the twisted-copper pairs of the public telephone and fiber optic networks, namely DLS systems and fiber access systems.

Since DSL use relatively high spectrum frequencies, its signal is susceptible to external noise sources. Thus, the research into new ways of reducing noise impact on network performance are extremely useful in terms of design of well established DSL systems (ADSL, ADSL2+) as well as in relation to latest generation (VDSL1, VDSL2) networks.

Crosstalk is the electromagnetic coupling that occurs when electrical signals are transmitted over telephone wires. It is the main factor limiting the bit rate and the distances that can be achieved on DSL systems. A pair of individually insulated twisted together conductors has been designed to reduce this coupling and to improve system performance. The reason for this is due to a sufficiently short space between twists - the electromagnetic coupling of energy over a small segment of wire is canceled by the out-of-phase energy coupled on the next segment of wire (Starr et al., 1999).

There are two kinds of crosstalk: Next (near-end crosstalk) and Fext (far-end crosstalk). Next is the main obstacle for systems that share the same upstream and downstream frequency

band. Next is the noise that appears on the other pair but at the same end of the cable as the source of interference (Cook et al., 1999), as shown in Fig. 1.

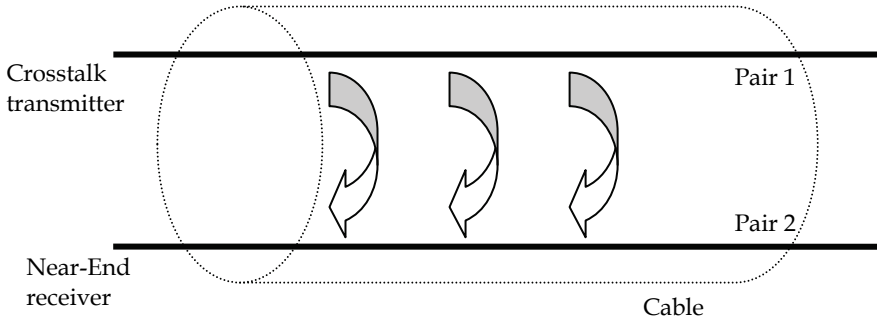


Fig. 1. Illustration of Next

Fext is the noise that appears on another pair, but at the opposite or far end of the cable to the source of noise (Cook et al., 1999). Fext is less harmful than Next since it is mitigated because the distance between the source and the noise receiver. Fig. 2 is an example of Fext.

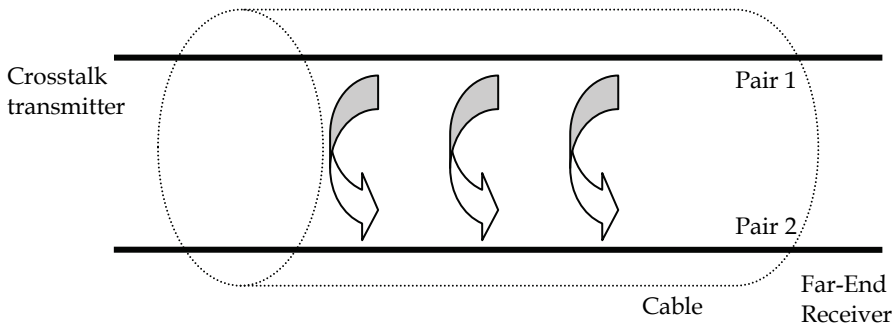


Fig. 2. Illustration of Fext

Techniques such as DSM (dynamic spectrum management) and MIMO (multiple-input multiple-output) schemes try to find a controlled injection of spectrum in DSL systems so that the resulting crosstalk can assume acceptable performance values (Starr et al., 2003), (Ödling et al., 2009).

2.2 Wireless Broadband Networks (WBN)

A large number of wireless technologies exist and other systems still being under design. These technologies can be distributed over different network families, based on a system scale (Nuaymi, 2007):

- A wireless personal area network (WPAN) is a data network used for communication among data devices close to one person;
- A wireless local area network (WLAN) is a data network used for communication among data devices: computer, telephones, printer and personal digital assistants (PDAs). This network covers a relatively small area, like a home, an office or a small campus (or part of a campus);

- A wireless metropolitan area network (WMAN) is a data network that may cover up to several kilometres, typically a large campus or a city;
- A wireless wide area network (WWAN) is a data network covering a wide geographical area, as big as the Planet. WWANs are based on the connection of WLANs, allowing users in one location to communicate with users in other locations.

There are many applications for wireless networks. One of the first uses for wireless technology was used as an alternative for traditional wired voice telephony, the narrowband wireless local-loop systems (Andrews et al., 2007). These systems, called wireless local-loop (WLL), were quite successful in developing countries whose high demand for basic telephone services could not be attended using the existing infrastructure. However, as conventional wired technologies such as DSL and cable modems began to be deployed, wireless systems had to evolve to support much higher speeds so that they could become competitive. A specific very high speed system called local multipoint distribution system (LMDS) was developed, capable of supporting several hundreds megabits per second in millimeter wave frequency bands, such as the 24 GHz and 39 GHz bands.

A WBN is a high data rate (of the order of Mbps) WMAN or WWAN. A WBN system can be seen as an evolution of WLL systems, mainly featuring significantly higher data rates. While WLL systems are mainly destined for voice communications and low data rate (i.e. smaller than 50 kbps), WBN systems are intended to deliver data flows in Mbps (Nuaymi, 2007).

There are a significant number of WBN systems with different and specific characteristics. Table 2 presents a comparison between the main WBN technologies (Andrews et al., 2007):

Parameter	Fixed WIMAX	Mobile WIMAX	HSPA	Wi-Fi
Meaning	Worldwide Interoperability for Microwave Access		High-Speed Packet Access	Wireless Fidelity
Standards	IEEE 802.16 - 2004	IEEE 802.16e - 2005	3GPP* release 6	IEEE 802.11 a/g/n
Frequency band	3.5 GHz and 5.8 GHz	2.3 GHz, 2.5 GHz, and 3.5 GHz	800/900/1,800/1,900/2,100 MHz	2.4 GHz and 5 GHz
Typical coverage	3-5 miles	< 2 miles	1-3 miles	< 100 ft indoors; < 1000 ft outdoors
Mobility	Not applicable	Mid	High	Low
Peak downlink (DL) data rate	9.4 Mbps in 3.5 MHz with 3:1 DL-to-UL ratio; 6.1 Mbps with 1:1	46 Mbps with 3:1 DL-to-UL ratio; 32 Mbps with 1:1	14.4 Mbps using all 15 codes; 7.2 Mbps with 10 codes	54 Mbps shared using 802.11 a/g; more than 100 Mbps peak layer 2 throughput using 802.11 n
Peak uplink (UL) data rate	3.3 Mbps in 3.5 MHz using 3:1 DL-to-UL ratio; 6.5 Mbps with 1:1	7 Mbps in 10 MHz using 3:1 DL-to-UL ratio; 4 Mbps using 1:1	1.4 Mbps initially; 5.8 Mbps later	

* Third-generation Partnership Project

Table 2. Comparison between main WBN technologies

Our focus in this section is to analyze WBN systems called pre-WIMAX systems. These systems use products which are claimed to be based on the IEEE 802.16 standard. They can deliver data flows up to 30 Mbps and their performance levels are close to the ones expected of WIMAX. Fig. 3 is a classical example of a pre-WIMAX system.

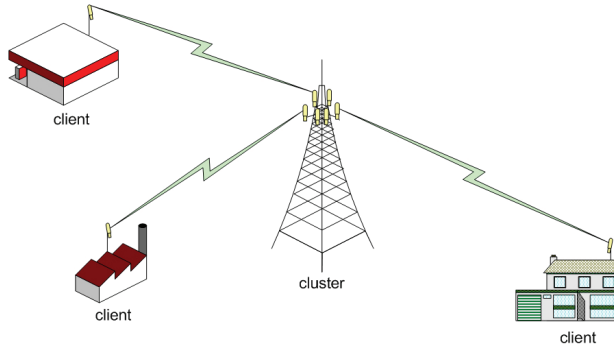


Fig. 3. Example of pre-WIMAX system

In this system we have a station server (or cluster) using six directional antennas (60° aperture) for an omni coverage. However, systems using 360°, 180°, 120° or 90° antenna apertures are also possible.

Pre-WIMAX systems can operate in the 2.4 GHz, 3.5 GHz, 4.9 GHz, 5.2 GHz and 5.8 GHz frequency bands. Depending on national regulation laws, pre-WIMAX systems can work in both licensed and license-exempt frequencies.

The main problem in pre-WIMAX systems is interference. Interference is an unwanted disturbance that can affect the overall system performance. Such disturbance is due to electromagnetic radiation emitted from diverse sources. It can appear in a different number of forms:

- Intra-system (within its own network, i.e., equipments working on the same frequency);
- Inter-system (external to its network, i.e., others systems working on the same frequency);
- External (other sources, not network but RF equipment, such as machinery and generators).

Traditional approaches to interference reduction include the use of power control, opportunistic spectrum access, intra and inter-base station interference cancellation, adaptive fractional frequency reuse, spatial antenna techniques such as MIMO and SDMA (space division multiple access), and adaptive beamforming, as well as recent innovations in decoding algorithms (Boudreau et al., 2009).

3. PLC applications across access networks

3.1 Using PLC on DSL systems

Consider the scenario of small or medium-size enterprise using a VDSL system (VDSL1 or VDSL2) as broadband access. In this system, the demand for higher data rates is increasing, especially when it uses services that require high bandwidth such as video conferencing and internet protocol television (IPTV). Thus, the proper control of crosstalk becomes a keystone in the operation of such systems.

Fig. 4 is a typical example of access network topology using VDSL systems on a fiber-to-the-curb (FTTC) scenario. A primary optical fiber cable connects the central office (CO) to a street cabinet, and from there, a copper pair is used to reach the customer premises equipment (CPE), i.e., the VDSL modem.

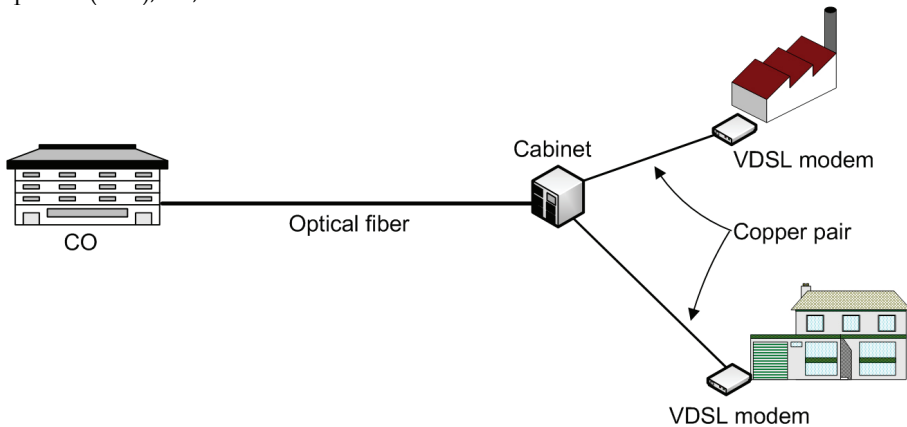


Fig. 4. Access network topology using DSL system on a FTTC scenario

VDSL is designed to operate over shorter loops. Consequently, VDSL equipment is positioned in cabinets, with the typical loop length being below one kilometer (Ödling et al., 2009).

A proposed use of the PLC is in the loop between the cabinet and VDSL modem. In this case, the PLC is used as a remote trigger for a system that changes the wires configuration on a telephone cable. The system shown in the Fig. 5 illustrates this use.

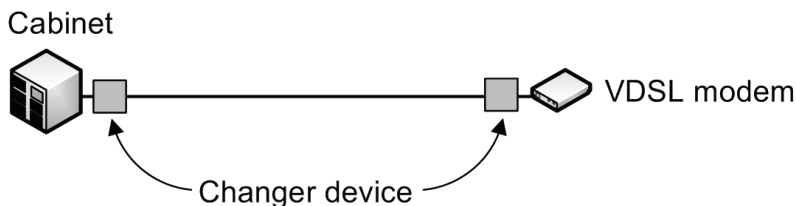


Fig. 5. Changer device using a PLC and a stepper motor

The changer device is comprised of a PLC and a stepper motor (an electromechanical system which converts electrical pulses into discrete mechanical movements). The main objective of this device is to modify the wire arrangement so that the resulting crosstalk has its values changed. It is obtained by changing the metal contacts located at the both extremities of the cable at the same time. This is the reason for it to be necessary to have two changer devices in the proposed configuration.

Obviously, this solution is a first approach method for reducing crosstalk impact, having a very specific application which is focused on heavy users who need a high quality transmission system with reasonable costs. A basic limitation of this proposed scenario is that it has no real use in a VDSL system using a single wire pair.

This scenario can be adapted to other DSL technologies. Fig. 6 shows an access network example for ADSL2+ technology.

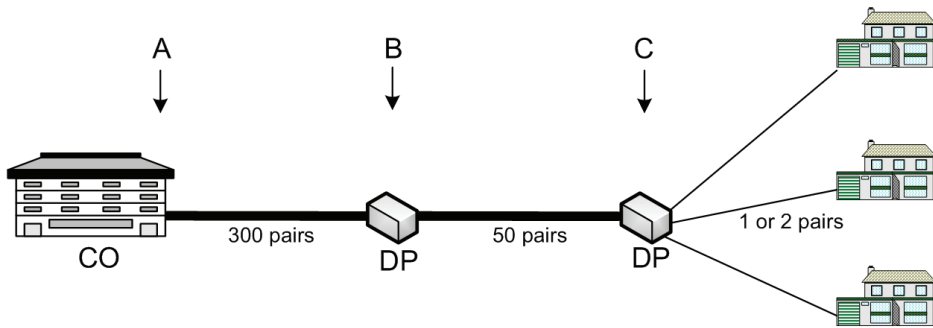


Fig. 6. Access network for ADSL2+ system

The copper plant is a star network which has fewer lines running together, until individual wire pairs finally reach their respective CPE (some configurations can use two wire pairs). Distribution points (DP) are the connection between cables of different gauges and wire numbers.

The changer device can be used between points A and B or between points B and C. The idea is the same as shown in Fig. 5, i.e., using the changer device to rearrange the layout of the metal contacts.

3.2 Using PLC on Wireless Broadband Networks (WBN)

The basic idea using PLC for interference reduction on WBN is to use it as an antenna azimuth automatic controller (AAAC).

Azimuth is the horizontal angular distance from the northern point of the horizon to a given referent direction. By changing the antenna's azimuth, the radiated power in a given direction is altered. As a result, it is possible to reduce the interference caused by frequency reuse within the same area of wireless coverage. In this utilization, the PLC is again used in conjunction with a stepper motor to perform the azimuth change.

The initial premise of this solution is to identify that interference is happening across the system. This can be done using some form of performance analysis system (depending on the equipment used, this could be a type of software for analyzing network performance) or collecting performance metrics from MIB (management information base) files, for instance. Once the occurrence of interference is identified, using the system described in Fig. 7, it is possible to perform a rapid and effective intervention on the system, thus reducing the interference effects.

Fig. 7 is an example of this proposed configuration. The PLC is connected to the stepper motor, which is responsible for the movement of set of APs (access points). AP represents the antenna of a radio transmission system. The number of APs will depend on the configuration of each system. The system shown in Fig. 7 uses six APs, where each antenna has a horizontal aperture of 60°. Others configurations, using horizontal apertures of 90°, 120° or other values are also possible.

The PLC control system consists of a computer (not shown in Fig. 7), which is responsible for sending commands to the PLC, thereby controlling the movements of the stepper motor. A basic ladder logic program for stepper motor control is shown in Fig. 8. In this case, i-TRiLOGI software (i-TRiLOGI, 2009) was used to perform an off-line simulation of the PLC's program on a personal computer.

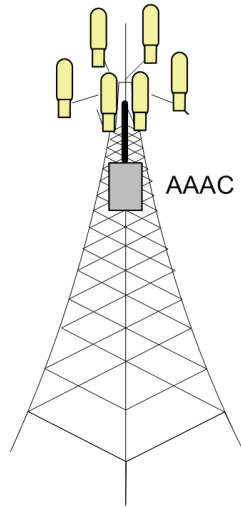


Fig. 7. Example of PLC application on WBN

```

I-TRIOGI Version 6.23 - (Educational Version)[C:\Documents and Settings\Lamarti...
File Edit Controller Simulate Circuit Help
Circuit # 1 1 2 3 4 5 Define Quick Tags Last

Stepper Motor Control Program

I. Statement: STEPSPEED ch, pps, accstep
-----
Purpose: Set Motion parameters for stepper ch#

parameters: ch = channel number (1-8)
pps = Max. output pulse rate in pulse/second (32-bit)
accstep = No. of steps to reach max. speed (1-32767)

Note: All three parameters may be integer variables, but if the
value exceed range then run time error may result.

II. Statement: a) STEPMOVE ch, count, rly#
b) STEPMOVEABS ch, position, rly#
-----
Purpose: a) Move stepper #ch by "count" steps.
b) Move stepper motor to absolute "position"

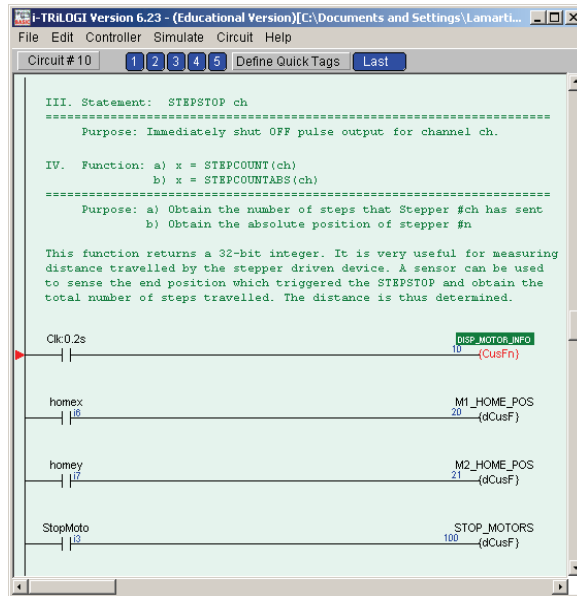
The absolute position is +/- value relative to a home position
which is set by the "STEPHOME ch" command.

rly# = The relay bit to affect when STEPMOVE is run.
It is turned OFF when run and ON when finished.
This can be used to signal the end of motion!

1st.Scan                               INIT
| |                                     1 (dCusF)
-----
Move1 | |11                             MOVE_FWD
| |                                     6 (dCusF)
-----
Move2 | |12                             MOVE_BWD
| |                                     4 (dCusF)
-----
PosIn1 | |14                             MOVEABS_POS1
| |                                     2 (dCusF)
-----
PosIn2 | |15                             MOVEABS_POS2
| |                                     3 (dCusF)

```

(a)



(b)

Fig. 8. Ladder logic program for stepper motor control: a) Code to control speed and movement, b) Code to control stop

4. Conclusion

We have presented alternative PLC applications on access networks, particularly in DSL systems and wireless broadband networks. Details about technical implementation possibilities are beyond the scope of this chapter; however the proposed applications use well known and easily accessible equipments and devices.

Since the PLC has relatively low cost, high operational speeds and multiple usage characteristics, its utilization across access networks provide a low-priced and practical method for mitigating problems related to the network performance.

5. References

- Starr, T.; Cioffi, J. M. & Silverman, P. J. (1999). *Understanding Digital Subscriber Line Technology*, Prentice Hall PTR, ISBN 978-0137805457, New Jersey
- Gonzalez, L. (2008). *DSL Technology Evolution*, Broadband Forum, http://www.broadband-forum.org/downloads/About_DSL.pdf
- Ödling, P.; Magesacher, T.; Höst, S.; Börjesson, P. O.; Berg, M.; Areizaga, E. (2009). The Fourth Generation Broadband Concept. *IEEE Communications Magazine*, Vol. 47, No. 1, January 2009, page numbers (63-69), ISSN 0163-6804
- Cook, J. W.; Kirkby, R. H.; Booth, M. G.; Foster, K. T.; Clarke, D. E. A. & Young, G. (1999). The Noise and Crosstalk Environment for ADSL and VDSL Systems. *IEEE*

- Communications Magazine*, Vol. 37, Issue 5, May 1999, page numbers (73-78), ISSN 0163-6804
- Starr, T.; Sorbara, M.; Cioffi, J. M. & Silverman, P. J. (2003). *DSL Advances*, Prentice Hall PTR, ISBN 978-0130938107, New Jersey
- Nuaymi, L. (2007). *WiMAX: Technology for Broadband Wireless Access*, John Wiley & Sons, ISBN 0-470-02808-4, West Sussex
- Andrews, J. G.; Ghosh, A. & Muhamed, R. (2007). *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Pearson Education, Inc., ISBN 0-13-222552-2, New Jersey
- Boudreau, G.; Panicker, J.; Guo, N.; Chang, R.; Wang, N.; Vrzic, S. (2009). Interference Coordination and Cancellation for 4G Networks. *IEEE Communications Magazine*, Vol. 47, No. 4, April 2009, page numbers (74-81), ISSN 0163-6804
- i-TRiLOGI 6.23 (2009). Educational Version, build 02, Triangle Research International, Inc, <http://www.tri-plc.com>

Development of Customized Distribution Automation System (DAS) for Secure Fault Isolation in Low Voltage Distribution System

M. M. Ahmed, W.L. Soo, M. A. M. Hanafiah and M. R. A. Ghani
University Technical Malaysia Melaka (UTeM)
Malaysia

1. Introduction

In general, an electric power system includes a generating subsystem, a transmission subsystem and a distribution subsystem. Electric power systems may have minor differences between countries due to geographical factors, demand variances, regions and other reasons. The voltages and frequencies for consumers around the world are depending on their regions. The power grids typically transmit electricity in three levels of voltage which are HV (100,000 Volts upwards), MV (1000 Volts to 100,000 Volts) and LV (1 to 1000 Volts). Fig. 1 shows the typical power production and distribution process.

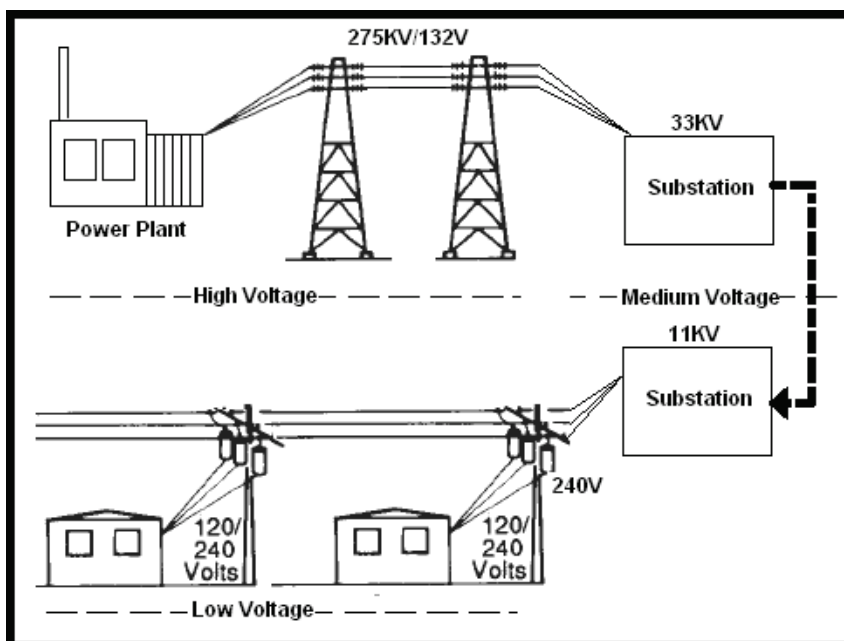


Fig. 1. Typical Power Production and Distribution Process

The electricity production process begins with its generation in power plants. The generated electric power is supplied through step-up transformers to raise the voltage to HV of transmission voltage before it is transmitted by transmission lines to transformer substations.

The substations reduce the transmission voltage via power transformer in Main Intake Distribution Substation (MIDS). MIDS is a node for terminating and reconfiguring transformers that step down the HV transmission voltage to Primary Distribution Voltage Level (PDVL).

The power is distributed from the transformer substations to the electric distribution network via Main Switch Station (MSS). Basically MSS is a node for terminating and reconfiguring the PDVL line of many feeders consisting of substations. In areas where power needs to be delivered to consumers, the power transformers in the substation are used to convert or step down the HV into a much lower voltage. Each feeder of MSS consists of a few substations that stepped down to consumer voltage. Basically, the network configuration for the distribution system is a loop circuit arrangement and each feeder consists of substations separated into two parts by the NOP.

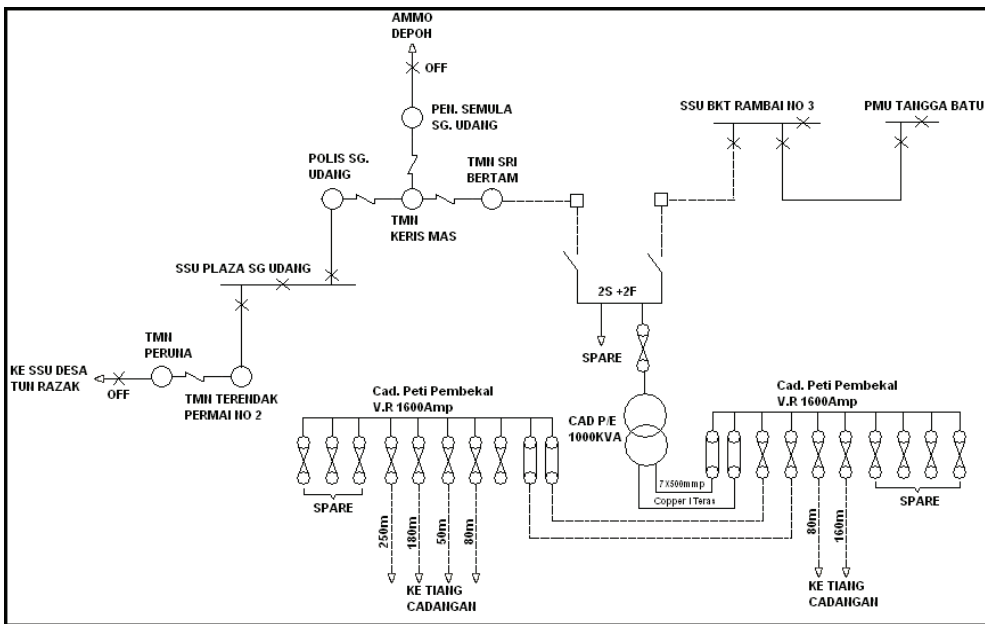


Fig. 2. An Example of Distribution Substation 11/0.415 kV

Most distribution systems are designed as either radial distribution system (Pabla, 2005) or loop distribution system. In some countries like Malaysia, the electrical connection of the substations is in the form of ring called "Ring (loop) Main Unit (RMU)". RMU can be obtained by arranging a primary loop, which provides power from two feeders. Any section of the feeder can be isolated without interruption, and primary faults are reduced in duration to the time required to locate a fault and do the necessary switching to restore service. The connections are illustrated in Fig. 3 and Fig. 4.

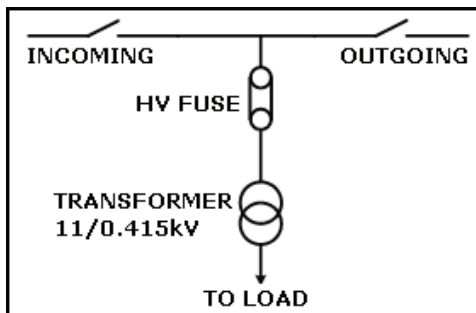


Fig. 3. RMU connection

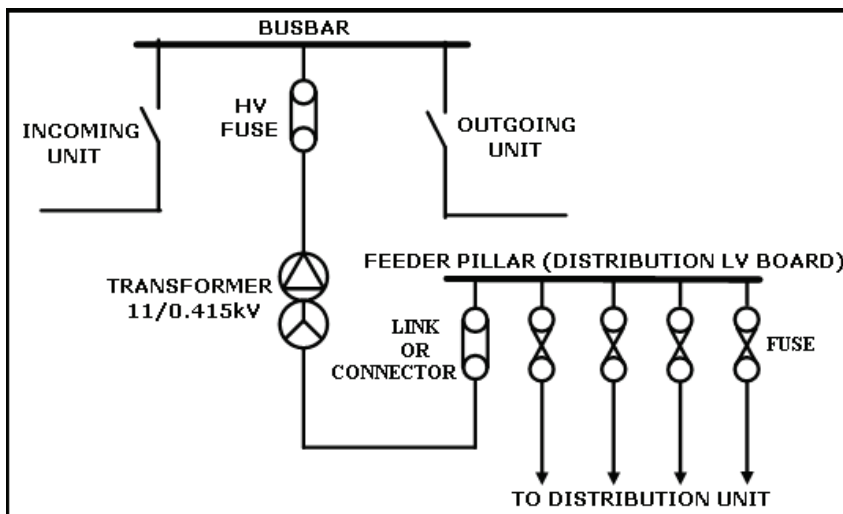


Fig. 4. Distribution Substation 11/0.415 kV

Substations serve as sources of energy supply for the local areas of distribution in which they are located. Their main functions are to receive energy transmitted at HV from the transmission lines, acted as nodal point from which the power or electricity can be changed or distributed from it to the other substations or consumers and provide facilities for switching. Substations are accessed by their incoming and outgoing switches connected by other substations and allow the fault point due to the substation which affects in the system that be isolated with switching method and the electricity remain supplied via other back up supply. They provide points where safety devices may be installed to disconnect circuits or equipment in the event of trouble. Some substations are equipped with EFI in order to locate the fault point either from upstream or downstream.

2. Low voltage distribution system

The low voltage operating equipment and systems are susceptible to faults, malfunctions and human errors. The solution to those problems lies on how the knowledgeable people such as engineers handle and solve them in the best possible ways.

The application of Automation system is one of the best solutions to those problems. In this book, an application of automation system has been proposed and described applied into practical LV systems for the solution of these problems.

However, the distribution systems have grown in an unplanned manner resulting in high system losses in addition to poor quality of supply. The other reasons are the lack of use of efficient tools for operational planning and advanced methodology for quick detection of fault, isolation of the faulty section and service restoration. Currently, fault detection, isolation and service restoration takes a long time causing the interruption of supply for a longer duration.

SCADA can be used to handle the tasks which are currently handled by the people and can reduce frequency of periodic visit of technical personal substantially. SCADA is a process control system that enables a site operator to monitor and control processes that are distributed among various remote sites. The control functions are related to switching operations, such as switching a capacitor, or reconfiguring feeders. Once the fault location has been analyzed, the automatic function for fault isolation and supply restoration is executed. When the faulty line section is encountered, it is isolated, and the remaining sections are energized. This function directly impacts the customers as well as the system reliability.

This research is to develop a state of the art technology which targets all types of LV systems and could be extended to lower voltage, medium voltage as well as higher voltage applications in electrical, electronic, communication and mechatronics engineering.

In the early stage of introduction, distribution control technologies have lagged behind if compared with advances in generation and transmission controls.

In Korea, the general structure of 154kV distribution substations using GIS standard. One distribution substation is composed of fixed devices such as a few transmission lines, 154kV double buses, two to four of 154kV/22.9kV main transformers, 22.9kV double structured distribution bus, many distribution lines, and switching devices like CBs and line switches (Lee & Park, 1996).

The fault point isolation is also based on the operation of corresponding relays and CBs but the switching operation is done manually by the operators. KEPCO has suggested four step processes to their operators. Step1 is to isolate the fault section using switches or CBs. Step 2 is to isolate black-out distribution line or transformers. Step 3 is to restore CBs one by one. The system uses radial operation and load transfer is allowed up to 90% of capacity of each transformer.

Bretas and Phadke (2003) proposed restoration scheme which composed of several Island Restoration Schemes(IRS). Each IRS is composed of two ANNs and a switching sequence program (SSP). The first ANN of each IRS is responsible for an island restoration load forecast. The input of this ANN will be a normalized vector composed of the pre-disturbance load. The second ANN of each IRS is responsible for the determination of the final island configuration and the associated forecast restoration load pick up percentage that will generate a feasible operational condition.

Hsu and Huang (1995) proposed ANN approach and pattern recognition method to provide a proper restoration plan in a very short period. They investigated service restoration following a fault on a distribution system within the service area of Taipei City District Office of Taiwan Power Company. In this paper, they concluded that the required Central Processing Unit (CPU) time using their method is much shorter than that required by the heuristic approach of reference.

Huang C.M (2003) addressed multi objective service restoration problem (SRP) with a fuzzy cause-effect network for minimizing a set of criteria, including the load not supplied and the number of switching operations. All of them are converted into a single objective function by giving relative weighting values for each criterion.

Hsiao et al (2000) proposed a reconfiguration for service restoration in a distribution system using a combination of fuzzy logic and genetic algorithms. The objectives of the proposed reconfiguration methodology were to maximize the load restored in the system and minimize the switching operations for the reconfiguration. However, the methodology proposed in this work is only applicable to radial power system.

3. Distribution automation system

The system architecture for this research is divided into three parts as shown in Fig. 5. The first part involves investigation of SCADA equipment or HMI. PC is equipped with GUI that runs under the Microsoft Windows XP platform using InduSoft software. The GUI provides monitoring for service substation and customer service substation, real-time data, data trending, data archiving, display and recoding alarm messages, show communication status of the system and control execution. Systems operations personnel use this equipment to control and monitor the I/O remotely.

Level 2 consists of I-7188EG embedded Ethernet. The control program is downloaded into the controller. The logic programming for service substation and customer service substation is almost identical. The logic programming is configured by using IsaGRAF software manufactured by ICPDAS. I-7188EG is responsible for communicating with the SCADA equipment using TCP/IP protocol. I-7188EG also acts as converter to link the SCADA equipment to the I-7044 module, I-7051 module and I-7042 module using RS485 protocol. The controller also receives data from power analyzer by using RS-485 protocol. Controller I-7188EG can handle control functions without the PC in real time.

Level 3 consists of I/O modules and three panels. The I/O modules are I-7044 module which is an 8 channel digital output and 4 channel digital input module, I-7051 which is a 16 channel digital input and I-7042 which is a 13 channel digital output. I-7044 module receives signal from ELCB in the customer service substation panel. It then converts the signal into RS485 standard signal and transfers to RS485 network. This signal is received by I-7188EG controller. I-7044 module receives signal from the controller to trigger certain actions to the relays as output devices. I-7051 and I-7042 are responsible to receive and send signals to I/O of service panel. Power analyzer is a power measurement metering device that displays volts, amps, watt, vars and etc. It sends data directly to the controllers to be displayed at the monitor.

In actual practice, service substation panel is connected to more than one customer service substation panel. In this research, service substation panel is only connected to one customer service substation panel. Customer service substation panel is connected to the consumer panel. In this case, the consumer panel consists of lights as the control loads.

Fig. 6 shows a typical compact substation (PE) which is still use until today. This compact substation fabricated by Schneider Electric Industries (M) Sdn Bhd. PE is also referred as RMU. A 12KV, 630A, 20KVA RMU is supplying power supply to LV Feeder Panel. A three-phase, 1000KVA, 11/0.433 kV transformer is used to step down 11kV to 433V before supplying to LVFP.

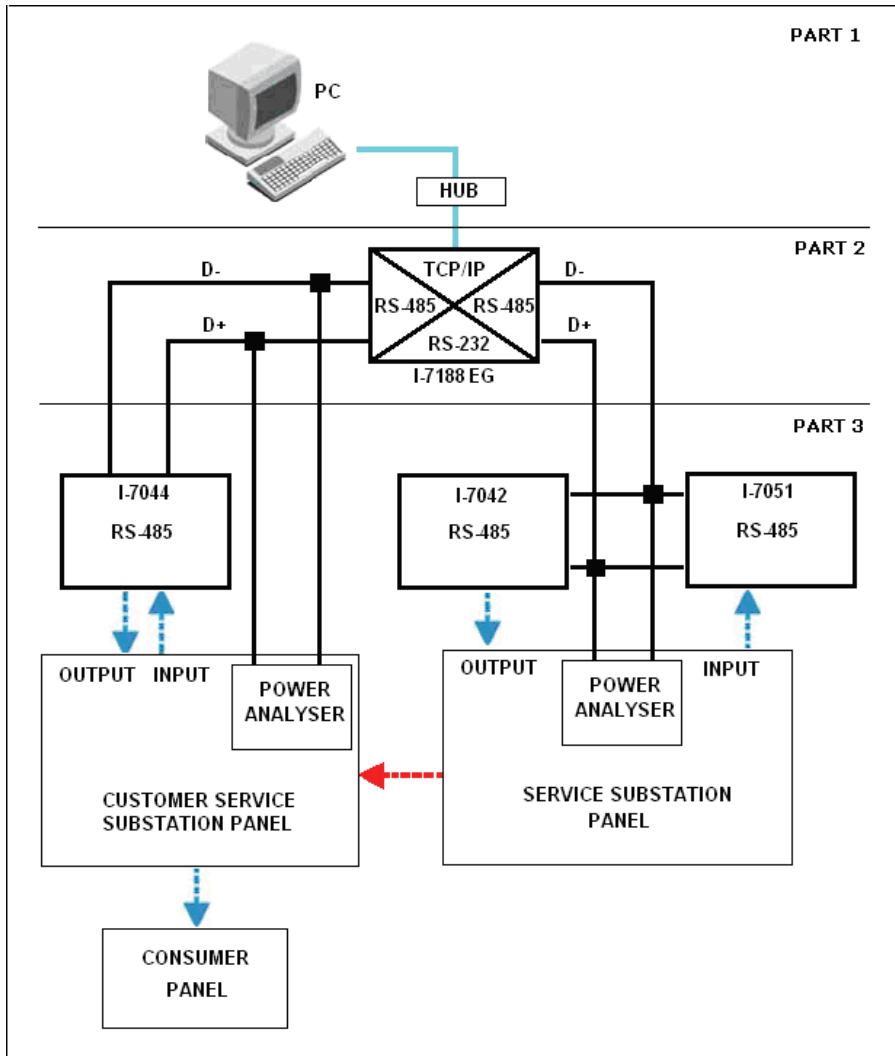


Fig. 5. System Architecture

The outgoing supplies are protected by fuses which have to be replaced if fault occurs. In this research project, fuses have been replaced by CB which can be manually or automatically controlled for switching operation and are not frequently replaced which is shown in Fig. 7. The panel in Fig. 6 is using power factor meter, kilowatt hour meter and three ammeters to provide reading of power factor, kilowatt hour and three phase current values. Instead of using different types of meters to provide the reading, a single power analyzer is used in this research to provide the same reading and is able to send the data to the controller using modbus protocol.

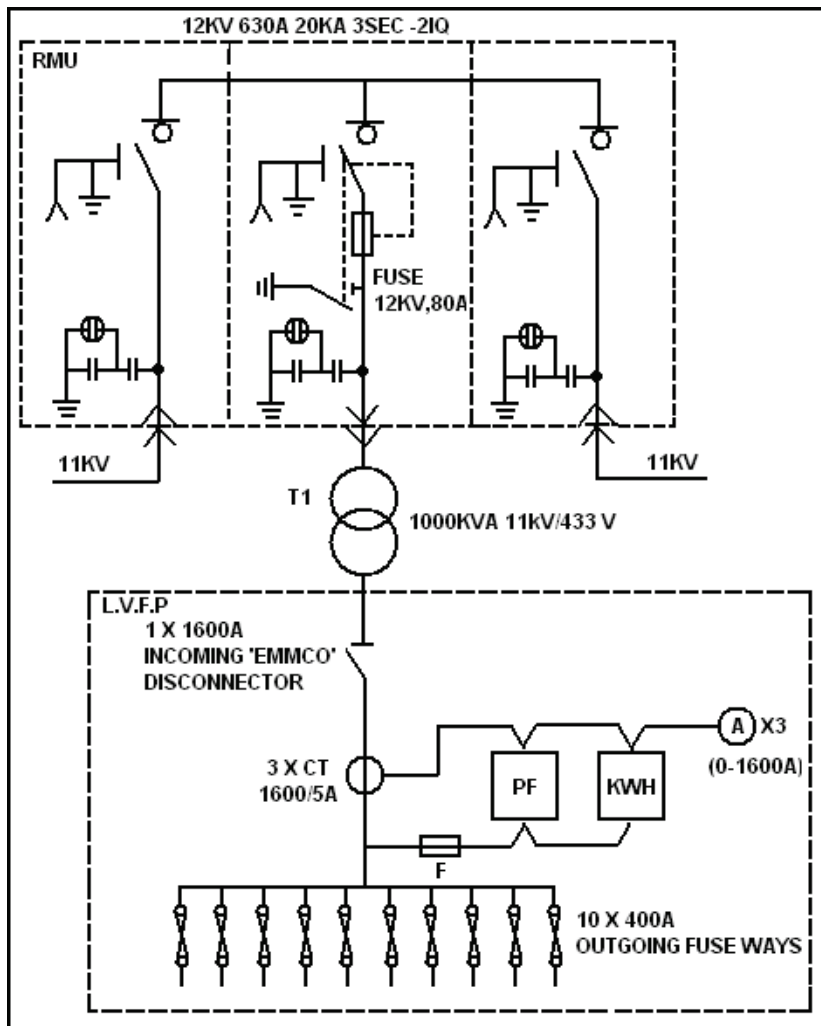


Fig. 6. Substation Installed By Schneider

5. HMI screen architecture

In Fig.8, the user interface starts with login screen. The default user name is “Guest”. The user needs to login before being able to use the toolbar described in Fig.9. The current user name will be displayed at the top right of the screen.

In Fig. 8, the control screen consists of main screen, customer service substation screen and service substation screen. These screens give flexibility on the operation personnel and also to assign authorization so that only the authorized personnel are being able to view the screen. The control screen can be divided into three sections which are the footer, header and body section. This is shown in Fig. 9.

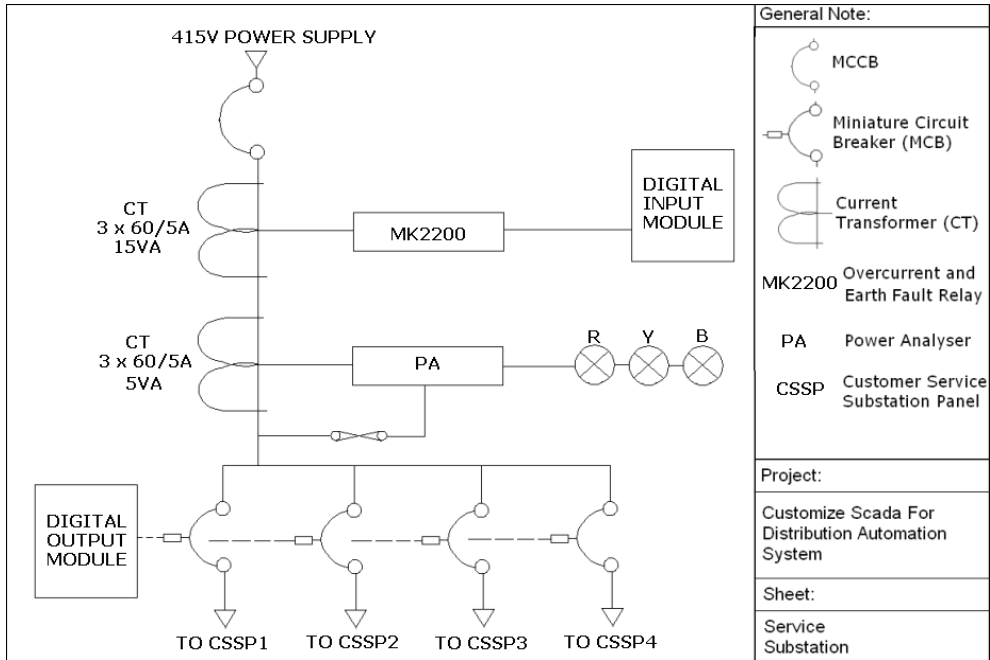


Fig. 7. Schematic Diagram of Service Substation

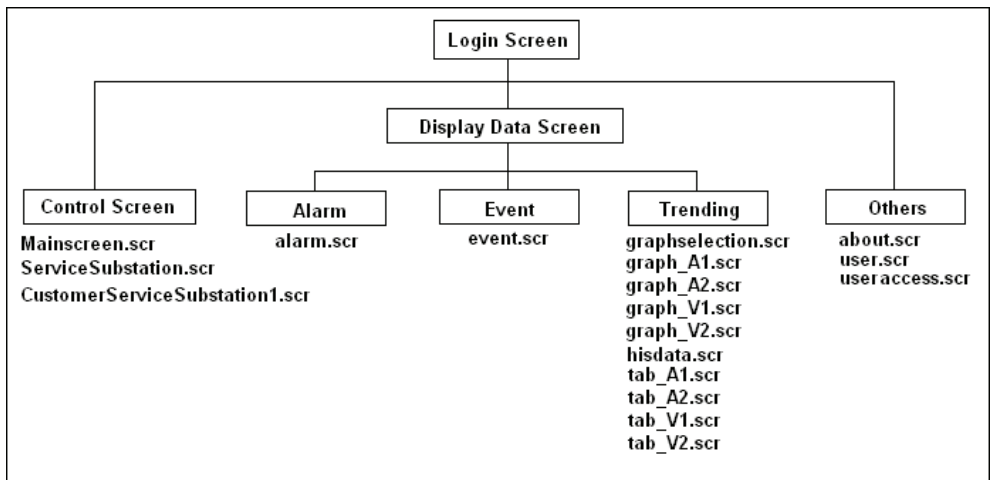


Fig. 8. Screen Architecture

The header section consists of toolbar, date and time display. Basically the toolbar provides navigation buttons to open other screens and also to exit the whole application. On the left side, the date and time display that provides the current date and time. The traffic light symbol on the top right indicates that the interface is communicating with the controller.

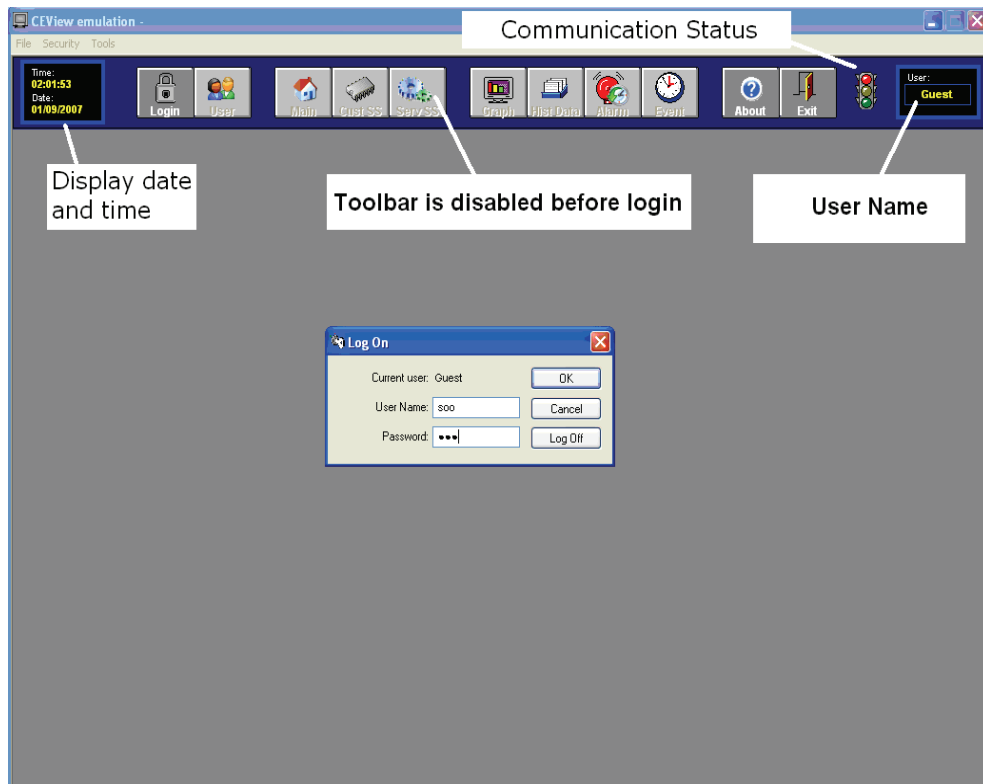


Fig. 9. Log on Dialog Box

The body section consists of symbols of I/O, control button, communication status, values of voltages and currents.

The footer section consists of alarm display screen that is located at the bottom of the main screen. Besides that the alarm display button is selected for filtering the alarm. Selection field is used to filter alarm messages by the selection text which is defined by the user. Priority field is used to filter alarm messages based on priority level. Total alarms will indicate the total alarm displayed. By pressing "Ack Top" button, the alarm on top of the alarm list will be acknowledged. Operation personnel can acknowledge all alarms by pressing on "Ack All". All the alarms displayed will be printed out once the print button is pressed. Operation personnel can press the "beep" button to disable the alarm's sound.

Fig. 10 shows the main screen of the SCADA Distribution Automation System. Main screen displays both panels which are service substation panel and customer service substation panel.

The symbols shown in Fig.10 are changed to red color to indicate false status and green color to indicate true status. The header part of the main screen consists of alarm screen, event screen, graph screen and table screen. Fig.11 shows the alarm screen. History alarms display alarm messages from the history database while online alarms display current alarm messages.

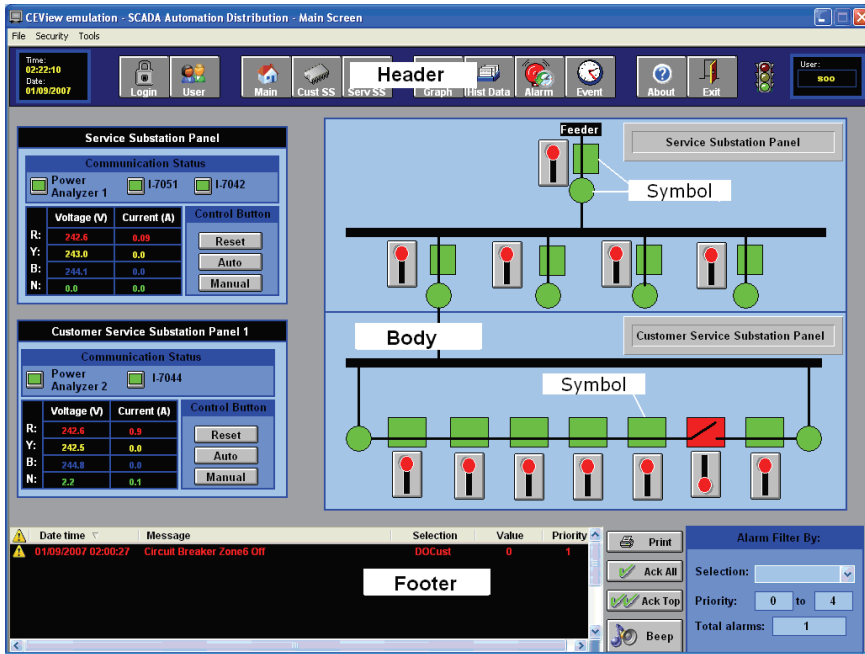


Fig. 10. The Main Screen



Fig. 11. Alarm Screen

Some tags are configured to the alarm parameters such as priority, selection, message, type and limit. Priority is to indicate the priority within the alarm group. Tags with a higher priority must have a higher priority value. Selection is used to filter in the alarm summary objects. The message part of the alarm screen is used to message the associated alarm that will be displayed on the Alarm/Event Control object. The type refers to the type of alarm such as Hi, Lo, HiHi and LoLo. Limit is the limit value associated to the alarm.

Event screen displays the list of event triggered by the operation personnel. For example, the event of pressing the reset button will be captured and displayed in event screen. Operation personnel can set specific date of event to be displayed or clicked on “today” button to display event that happened today. Fig.12 shows the event screen.

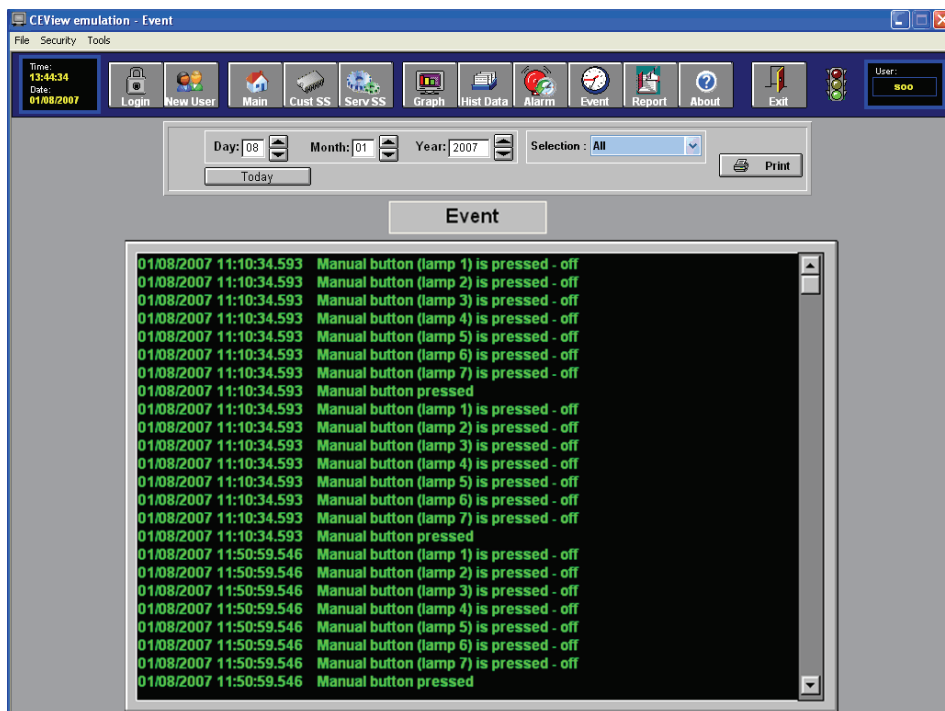


Fig. 12. Event Screen

The graph will provide on-line data of voltage and current from the power analyzer. Cursor button located at the top of the graph can be dragged to know the value along a certain position of the graph. X-axis and Y-axis can be adjusted to the user preferences. A combo box on the left side of the graph is used to change the property of X-axis and Y-axis. Start Date, Start Time and duration are used to set the X-axis. Fig.13 describes the trending screen for the customer service substation panel to display voltage values.

By clicking on the “Show Tabular” button, screen shown in Fig.14 will be displayed. The voltage reading displayed in the table shown in Fig.14 is captured from the voltage graph in Fig.13. “Reload” button is to reload the voltage reading from the voltage graph into the table.

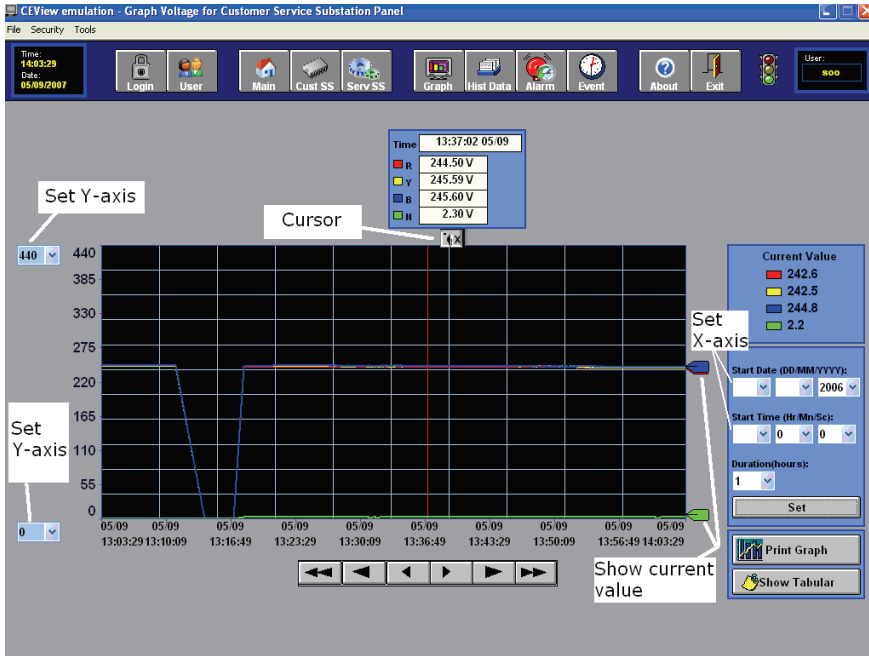


Fig. 13. Voltage Graph for Customer Service Substation Panel

The screenshot shows a software interface for monitoring a substation panel. The main window is titled "CEView emulation - Data Table for Customer Service Substation Panel". It features a toolbar with icons for Login, User, Main, Cust SS, Serv SS, Graph, Hist Data, Alarm, Event, About, and Exit. The user is identified as "SOO".

The central part of the interface displays a table titled "Table of Voltage Data For Customer Service Substation Panel". The table has the following columns: ID, Date, Time, Voltage,R, Voltage,Y, Voltage,B, and Voltage,N. The data is as follows:

ID	Date	Time	Voltage,R	Voltage,Y	Voltage,B	Voltage,N
1	03/05/2007	12:49:00	232.200000	10.000000	0.000000	2.100000
2	03/05/2007	12:50:00	232.300000	10.100000	0.000000	2.100000
3	03/05/2007	12:53:00	232.200000	10.100000	0.000000	2.100000
4	03/05/2007	12:54:00	232.100000	10.100000	0.000000	2.100000
5	03/05/2007	13:05:00	230.900000	10.100000	0.000000	2.100000
6	03/05/2007	13:06:00	231.200000	10.100000	0.000000	2.100000
7	03/05/2007	13:07:00	231.800000	10.200000	0.000000	2.100000
8	03/05/2007	13:18:00	230.700000	10.100000	0.000000	2.100000
9	03/05/2007	13:19:00	230.300000	10.200000	0.000000	2.100000
10	03/05/2007	13:22:00	230.300000	10.000000	0.000000	2.100000
11	03/05/2007	13:25:00	230.200000	10.100000	0.000000	2.100000
12	03/05/2007	13:36:00	229.700000	10.200000	0.000000	2.100000
13	03/05/2007	13:40:00	229.900000	10.000000	0.000000	2.100000
14	03/05/2007	13:41:00	229.600000	10.100000	0.000000	2.000000
15	03/05/2007	13:42:00	229.400000	10.200000	0.000000	2.000000
16	03/05/2007	13:43:00	229.300000	10.200000	0.000000	2.000000
17	03/05/2007	13:44:00	229.300000	10.100000	0.000000	2.000000
18	03/05/2007	13:45:00	229.500000	10.100000	0.000000	2.000000
19	03/05/2007	13:46:00	229.700000	10.200000	0.000000	2.100000
20	03/05/2007	13:47:00	229.200000	10.100000	0.000000	2.000000
21	03/05/2007	13:48:00	229.200000	10.400000	0.000000	2.000000

On the right side, there are buttons for "Reload" and "Print".

Fig. 14. Voltage Data Table for Customer Service Substation Panel

User dialog box is to create new user, delete user, block and unblock user. Fig.15 shows the user dialog box. When “New User/Delete User” button is pressed, the screen shown in Fig.16 will be displayed. User can be blocked from accessing the application by clicking on “Block User” button. “Unblock User” button will undo the block user action. User can be removed by clicking on “Remove User”. “User Access” is to set which screens that are allowed to be accessed under certain user group. Fig.21 illustrates the user access dialog box.

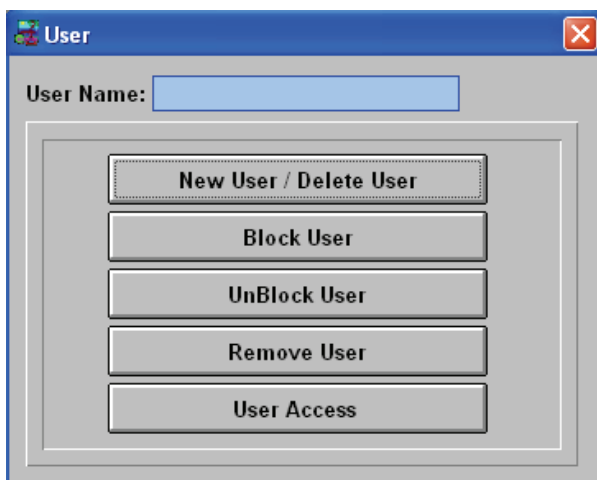


Fig. 15. User Dialog Box

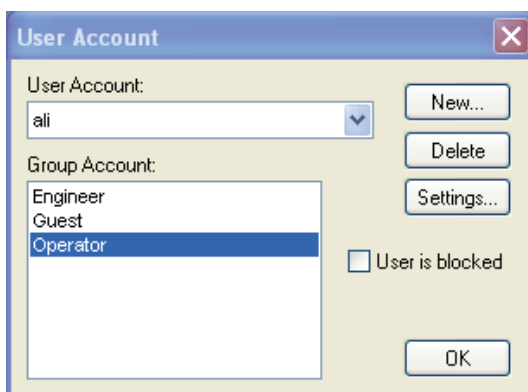


Fig. 16. User Account Dialog Box

“New” button is used to create new user and a dialog box as shown in Fig. 19 will be displayed. “Delete” button is to delete user. A confirmation message box will be displayed to confirm the deleted action. “Setting” button is to change the password of the selected user as shown in Fig. 20. User can also be blocked by checking the check box which indicates that this “user is blocked”.



Fig. 19. New User Account Dialog Box

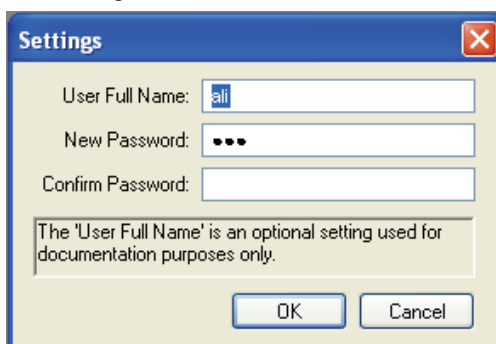


Fig. 20. Setting Dialog Box

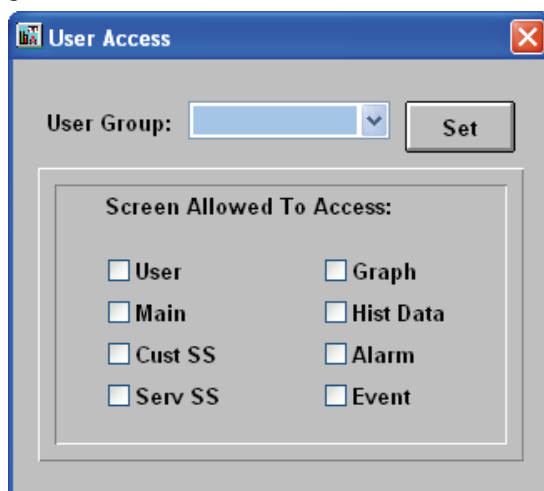


Fig. 21. User Access Dialog Box

6. HMI screen architecture

6.1 System operation and experimental

The system can be set to manual mode or automatic mode. The two operations which are the Operation Without Fault and Operation With Fault are described below.

6.2 Operation Without Fault

After success login to the system, control screen to be displayed is chosen from the toolbar. An example is shown in Fig.21, when “Serv SS” button is pressed, the service substation screen will be displayed. Currently all outputs are in healthy conditions. The communication status shows no communication error with the modules as well as the power analyzer. Fig.22 shows the values of counters during normal conditions. Loads will be turned in sequence and ‘count_L_cur_c’ will be increased by one.

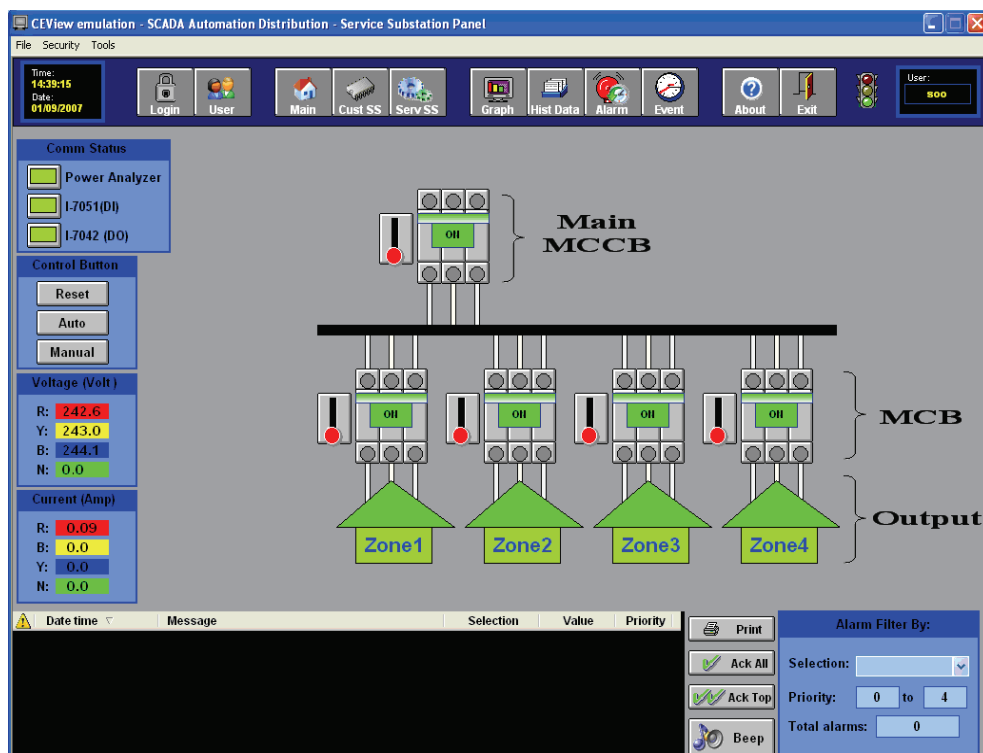


Fig. 21. Healthy Condition at the Service Substation Panel

6.3 Operation With Fault

If fault occurs, the ELCB will detect the fault condition and terminate all supplies to the zones. The outputs are the zones consisting of loads. In Fig.23 main MCCB, MCBs and the outputs are turned off. Alarms will be triggered and displayed at the bottom of the screen.

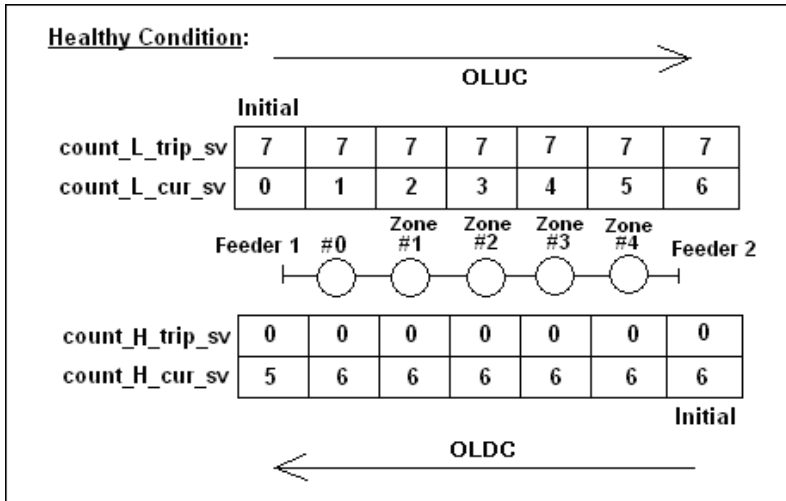


Fig. 22. Counters for OLUC and OLDC

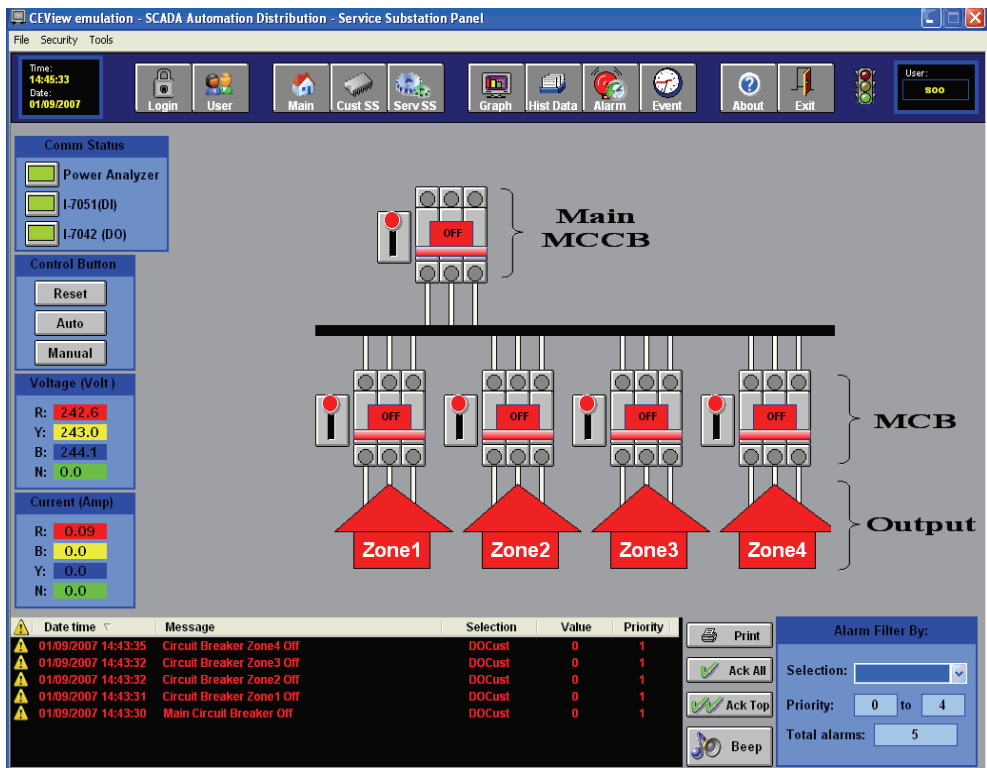


Fig. 23. Unhealthy Condition at the Service Substation Panel

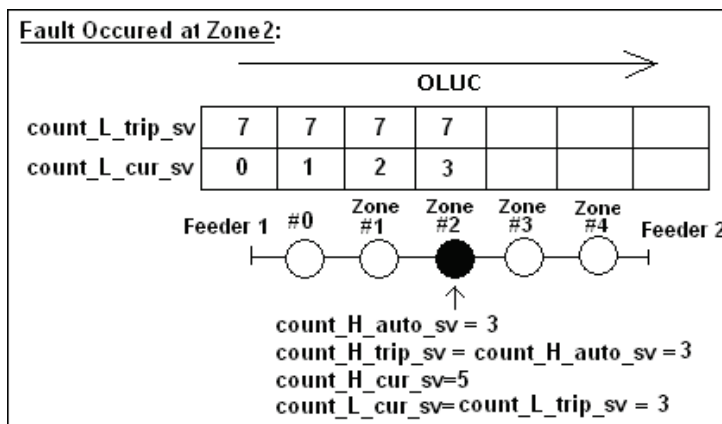


Fig. 24. Fault has occurred at Load 2

If the system is operated under automatic mode, the controller will automatically reset the ELCB using a delay timer and the power input will be turned on again. OLUC will execute where each load will be turned on in sequence starting from main MCCB or load 0. Trip will occur once the faulty load is turned on and the 'count_L_cur_sv' indicates this faulty load which is described in Fig.24.

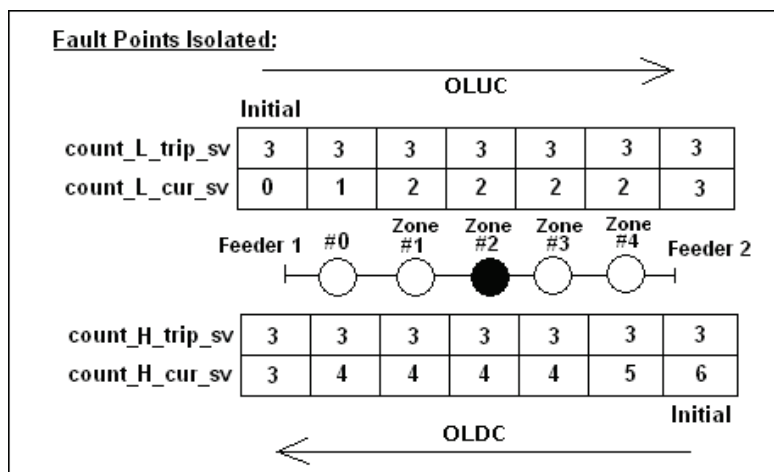


Fig. 25. Fault Point has been Isolated

The ELCB is reset and electricity power supply is supply to the system. OLUC will be executed once again and will turn on main MCCB and Zone1. However Zone2, Zone3 and zone4 are still remained off. Then the OLDC will be executed. The OLDC will turn on Zone 3 and Zone 4. Fig.26 shows Zone 2 which is the fault zone will remain off.

In the alarm list, the blue color indicates that the output has changed to healthy status and the red color indicates that the output is still remained unhealthy. Once the fault points have been checked and repaired, the "Reset" button is pressed to reset the counter and the OLUC and OLDC will be executed again and at this time, zone 2 will be turned on.

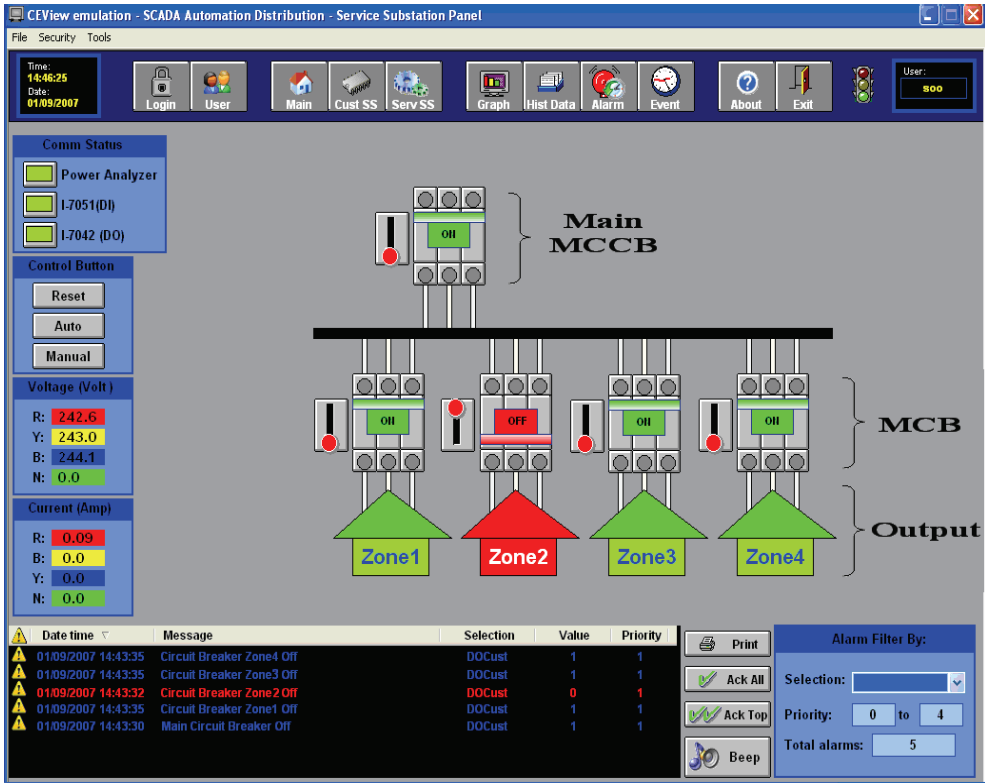


Fig. 26. Fault Isolation at the Service Substation Panel

7. Conclusion

In this research project, a Customized SCADA based RTU for service substation and customer service substation is developed by using the open loop concept for the distribution networks. Currently, the SCADA system in the low distribution system implemented by TNB only focuses on alarm monitoring. SSO has to operate the control functions at the HMI side. The operator needs to analyze the situation and to make appropriate actions.

In this research, a Customized SCADA is built to provide automatic fault isolation for low distribution system. The objective of the research is to provide an HMI for SCADA that is capable to build interfaces with I/O devices and fault isolation method. The contribution of this research includes developing a complete fault isolation algorithm based on an open loop distribution system. Service Substation Panel, Customer Service Substation Panel and Customer Panel have been built to validate the proposed methodology. In this chapter, the proposed methodology is summarized with the experimental results and conclusion based on the results is also highlighted.

The HMI is capable to communicate with the I/O devices. An HMI for SCADA is developed in this research by using an embedded Ethernet controller as the converter to communicate with the I/O devices. In addition to the automation, the controller retrieves the data from

the power analyzer. The SCADA software used in this research is propriety software shared by the same protocol with the embedded Ethernet controller which makes the communication possible by using modbus TCP protocol. The HMI can be monitored at different sites as the controller equipped with TCP/IP features. Whenever the system detects fault, an alarm message will be displayed at the HMI side to acknowledge the operator. The status of communication between the controller, digital I/O modules, power analyzer with the HMI helps to acknowledge the operator if there is a communication breakdown. This feature improves the reliability of the SCADA system in sending or receiving data from the devices.

Based on the experimental results, the system correctly locates the fault point, isolates the fault point and reenergizes the un-faulted loads. However, during the fault isolation operation, the system has to detect the fault point by simply switching on the fault load. After the system acknowledges the fault point, the appropriate switching functions are executed. The developed system has a potential in reducing the outage time while comparing to the manual operation by the technicians and engineers.

The SCADA system provides GUI, alarm system, data logging and report management facilities for the operator to interact with the equipment in the service substation and the customer service substation. The usage of RTU allows for future expansion. New technologies such as GSM and satellite communication are becoming available and will allow for new low cost automation system solutions.

8. Future research

This research project is a step towards developing customized DAS for LV distribution system and much more work lies ahead in evaluating new DAS technology and in developing new DAS applications. However, the outcome of this research as first effort has been very encouraging and bringing beneficial to the DAS research development in the future.

- a. Using web based application as GUI supported by IP-based computer networks and internet connectivity as communication network to remote sites.
- b. Improving the data communication system for distribution automation such as Global System for Mobile (GSM), Code Division Multiple Access (CDMA) and Wireless in Local Loop (WLL), Distribution Line Carrier Communication (DLCC) and Radio Communication.

9. References

- Lee, H.J. & Park, Y.M. (1996). A Restoration Aid Expert System for Distribution Substations. IEEE Transactions on Power Delivery, Vol. 11, pp.1765 -1769
- Bretas, A.S & Phadke, A.G. (2003). Artificial Neural Networks in Power System Restoration. IEEE Transactions on Power Delivery, Vol.18, No.4 , pp.1181 -1186.
- Hsu, Y.Y & Huang, H.M. (1995). Distribution System Service Restoration using the Artificial Neural Network Approach and Pattern Recognition Method. IEEE Proceeding Generation Transmission Distribution, Vol. 142, No.3, pp.251-256.
- Hsiao, Y. T. & Chien, C. Y. (2000). Enhancement of Restoration Service in Distribution Systems using a Combination Fuzzy-GA Method. IEEE Transactions on Power Systems, vol. 15, no. 4, November 2000, pp. 1394-1400

Huang C.M (2003). Multiobjective Service Restoration of Distribution Systems Using Fuzzy Cause-Effect Networks. IEEE Transactions on Power Systems, Vol. 18, No. 2, pp.867 - 874

Computer Emulations to Support Training in Automation

Manuel E. Macías and Ernesto D. Guridi
*Electrical and Computing Engineering, ITESM Campus Monterrey
Monterrey N. L.,
México*

1. Introduction

With the main objectives of complementing the process rather than eliminating the real experience from the student, and making it more efficient, the Electrical and Computing Engineering Department at ITESM Campus Monterrey has been developing and using, for the last 3 years, computer emulations of real industrial processes at the Tele-Engineering Lab. The obtained results have shown that the students perform much better than in a traditional laboratory session.

An Emulation is a computer model that mimics the operation of a real or proposed system. With Emulations, different solutions can be implemented and tested without the availability of the real system. Emulations are goal directed experimentation using dynamic models. Hence, it provides repeatable experimentation opportunities under controlled and extreme conditions [1-3]; students are able to experience several possible problems before facing them in real systems. As it is expected and even desired, students make mistakes while programming. A serious mistake on a computer screen is infinitely preferable to a mistake in a real system.

Another important aspect to consider is that every single student has his/her own learning method and also his own rhythm. One cannot expect to get the same results, in the same manner, from two different students [4, 5]. It is important to keep in mind that there may be several solutions for the same assignment and that the complexity between different solutions may vary. When students are restricted to obtain a solution in a certain period of time within a laboratory session, their chances to reach a correct solution could be limited. By handing the emulations to the students, the laboratory concept is extended. The students are then able to program, test, and debug their PLC programs without being restricted to a scheduled laboratory session. The laboratory, is therefore, virtually always available.

2. Main advantages

The aims of using emulations are, respectively, gaining insight, performance prediction, and finding the appropriate input values for a desired behavior [3].

Therefore, Emulations are useful to support education and provide training in areas where dynamic systems are involved [1, 2, 5, 6]. Concerning the Industrial Automation field, there are several advantages in having an emulated system:

- **Costs:** One of the most noticeable benefits is the cost. Many universities cannot afford enough appropriate equipment, such as scaled models of industrial processes that help them to develop practical skills on the students. With the use of Emulations, universities could acquire a single model to equip their laboratories or even, if their budget is not enough, make the students practice by the only use of Emulations.
- **Debugging:** Programming errors discovered using Emulations would not be as costly, because the errors discovered during the testing process would not damage the equipment. The use of the emulations to debug the student's program highly reduces the possibilities to commit errors in the real system.
- **Availability:** In most cases, only one actual system would be available and students would only be able to test programs on the system one at a time, making the process inefficient. This is not the case with the Emulation, where students can be testing programs simultaneously, because of the fact that each student could have his own virtual duplicate of the system at his own computer.
- **Animation:** Using *animation* to visualize system behavior greatly increases the ability to spot problems and certainly enhances students' learning [2, 7]. The process behaves accordingly to the student written code.
- **Diversity:** With diverse kinds of processes, the students can practice different programming techniques, enriching their qualifications. A University could have a library of emulated models, containing processes with different complexities, each of them to be used in homeworks, projects, or tests.
- **Versatility:** With Emulations you can quickly try out your ideas. Any promising correct solution is either accepted or dismissed much faster.
- **Overall understanding:** Students have a better overall understanding of a new system or process when they work with its Emulation. Sometimes sensors in real systems are not accessible or visible to the student. In the Emulation, the location of each sensor is clearly specified.
- **Handling of Time.** An often named advantage of emulations is that they (virtually) instantly show the results of the students' decision. Furthermore, the "expansion of time" is also a major characteristic of emulators; users have more time than in reality to contemplate a complex situation and to make a decision. Emulators with adjustable time frames can be used [4].

3. Process Emulations' characteristics

Although we know that Emulations are not always a perfect match of the physical system behavior, it is attempted with them to virtually duplicate a real system.

As a matter of fact, there are certain process characteristics that, because of their rare or occasional existence (such as hardware failures, external disturbances, or even a behavioral modification), cannot be exactly replicated.

Therefore, Emulations should be *as similar as possible* to the actual process. The operational characteristics of the sensors and the actuators must be preserved, such as the signals provided by the sensors and the required signals to activate the actuators. Also, it is desired that every sensor can be manipulated with a mouse click. This allows the user to activate the sensors at any time, the same way it happens in the real system using an external object.

Through a process of abstraction, we select those details that are most critical to characterize the operation of the system. The degree of detail required in a model depends on the nature of the system itself.

An important aspect to consider when developing Emulations is the appropriate selection of the view to be used, in order to have a complete visualization of the process. It is desired that any possible programming error can be detected. Also, even when it is almost impossible to predict all the probable errors that could appear while programming, the behavior of the Emulation before common error situations can be established. For instance, if an actuator used to move an object is not deactivated when the object has reached its physical limit, in the real system the actuator will be forced and the object will not move beyond this limit; in the Emulation the object will continue moving off the screen, helping the student to detect his error.

3.1 Real processes: scale models

At the Tele-Engineering Lab several scale models of industrial processes are currently being used, such as: Transport and Sorting Lines, Process Lines with Machine Tools and 3-Axis Portals, from the Staudinger GmbH company.

The Transport and Sorting Line model, shown in Figure 1, simulates a handling device to allocate part loads from a store register to various discharge stations, as used, for example, in a parcel distributor's logistics.

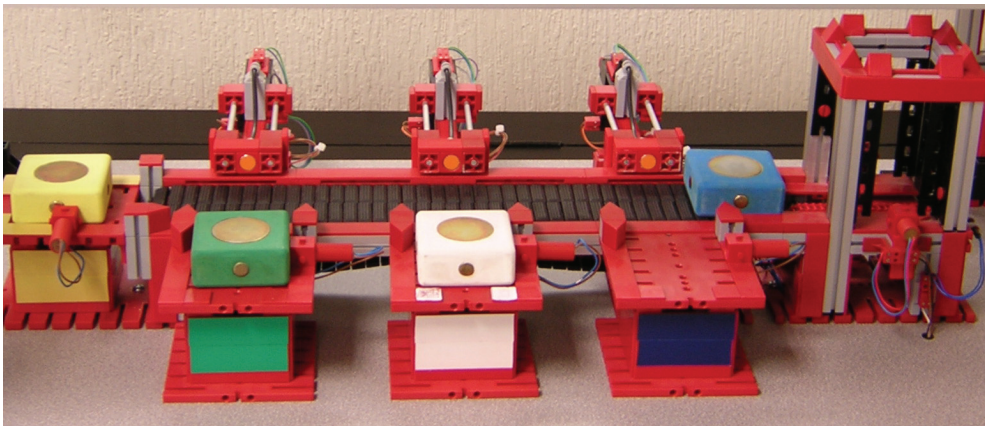


Fig. 1. Scale Model Transport and Sorting Line

The Transport and Sorting Line consists of a storage register with an integrated conveyor chain, a conveyor belt, three pushers, and four discharge stations. The scale model shows parcels being withdrawn from the store register, being recognized at an identification unit, getting transported to the corresponding discharge station by a conveyor belt, and finally being poked from the conveyor belt to the discharge station by a pusher.

3.2 Emulated processes

Each of the real models used at the Lab has been already emulated. The Emulations of the models were created using LabVIEW from National Instrument as the development tool, since it offers a graphical interface to the user, who can easily operate it.

The emulated model of the Transport and Sorting Line is shown in Figure 2. As it can be seen, a top view was selected to emulate this system. The reason of this is that this view

provides a complete visualization of the process. Also, the location of each Sensor is clearly identified.

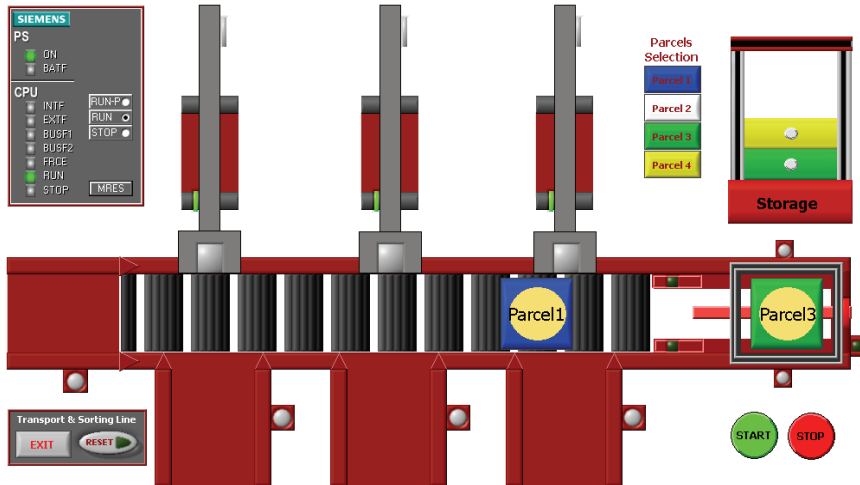


Fig. 2. Emulated Model Transport and Sorting Line

The scale model of a Process Line with Machine tool and its corresponding Emulation are shown in Figures 3 and 4, respectively. The process line consists of a turret drilling machine, three conveyor belts, a slewing table with conveyor chains and an automated lay-in-unit. The scale model shows a workpiece being provided at the lay-in-unit, then being brought onto the conveyance by a pusher, getting transported to the process cell, getting machined" in several steps and finally being brought out to a discharge station by using the slewing table.

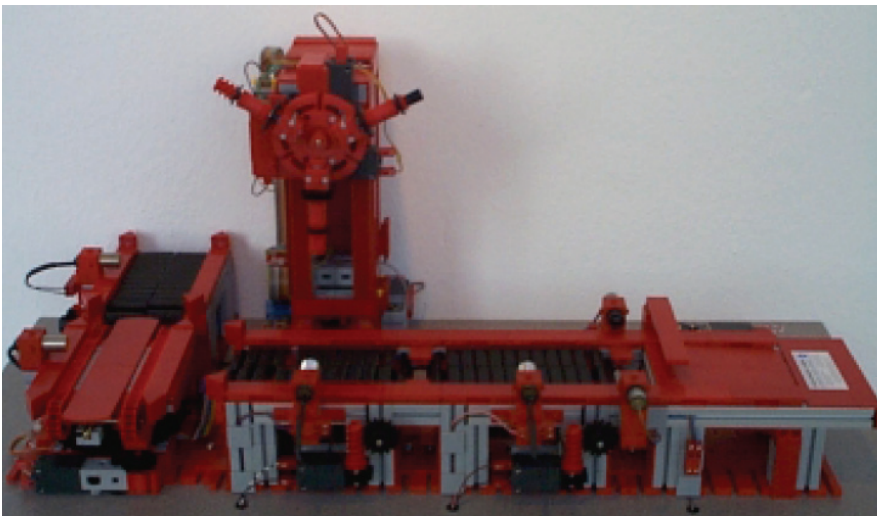


Fig. 3. Scale Model Process Line with Machine Tool

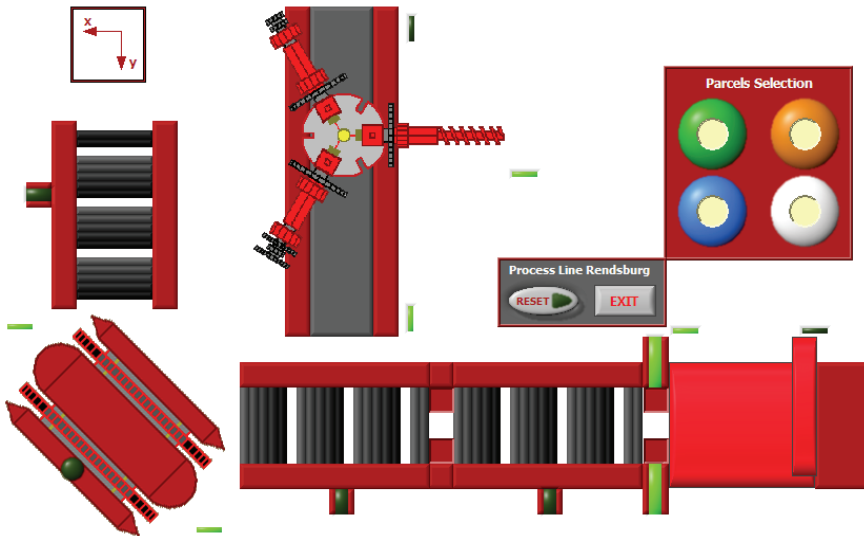


Fig. 4. Emulated Model Process Line with Machine Tool

At the present, a diverse library of Third-Dimension (3D) Emulations is being developed. 3D Emulations offer, as it could be expected, more characteristics than the ones in Two Dimensions (2D). In a 3D visualization it is possible to get immersed into the process and actually adapt the view to observe specific tasks or circumstances occurring during the process. On a 3D environment the student can zoom, rotate, or scroll the process at will (see Figure 5).

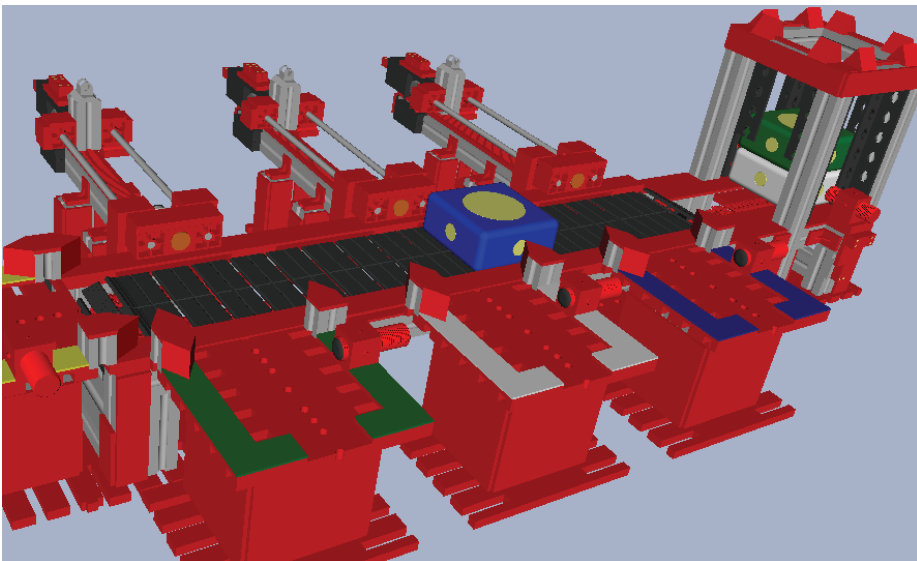


Fig. 5. Transport and Sorting Line Emulated on a 3D Environment

Taking advantage of the 3D visualization, students can observe the process from perspectives that most of the times are not available on real systems (see Figure 6).

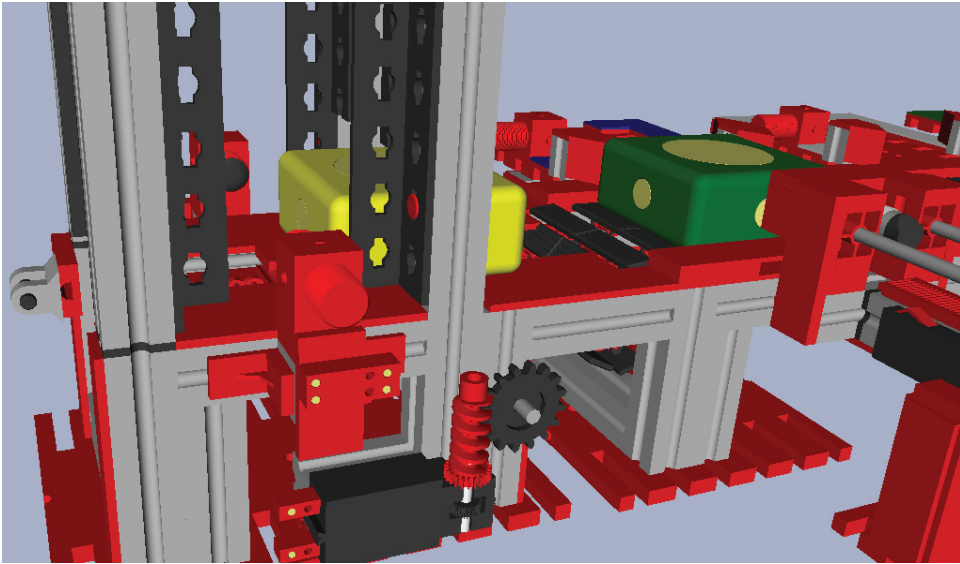


Fig. 6. Different Perspective on the 3D Environment

4. Use of the Emulations

Before the development of the Emulations, the students had to solve the exercises and the Automation projects at the laboratory, and only within the laboratory sessions. This caused that pretty often, students could not finish the exercises on time, since the automation tasks normally require a considerable amount of tests in order to get to a correct solution. If the students wanted to prepare the exercises before the laboratory sessions, they could only sketch on paper a possible solution and they did not have the possibility to validate it. Most of the times, the proposed solution contained programming errors that needed to be debugged and several tests were required.

This fact forced that either the exercises had to be designed in such way that they could be solved in a laboratory session, reducing their complexity, or that the same exercise had to be solved using different sessions. In both cases, the process was inefficient and the desired abilities were not developed in the students as desired.

This is not the case when using Emulations. By handing the emulations to the students, they do not have to wait until the next laboratory session in order to validate their proposed solution. Therefore, the students have more flexibility to test and debug their PLC programs, because they can work wherever they want and without being restricted to a schedule.

For this purpose, the students receive the tools required to solve the different assignments outside of the laboratory session: the stand-alone executable version of the determined Emulation that they need to automate and the Student Version of the development Software, Step7, from Siemens, which also includes the computer simulation of the Siemens controllers, S7-PLCSIM.

Students are given an automation task to solve, in which they have to make an analysis of the process in order to gain ideas that approach them to a correct solution. The implementation of their proposed solution may be unsuccessful and a redesign will be required. This leads the students to try their solutions on the Emulation several times in order to get to the final one. If the laboratory session comes to an end before they have reached a correct solution for the task, they can continue working outside of the laboratory. At the next laboratory session, the instructor tests the PLC program in the Emulation and only when the program runs completely out of errors, the students are asked to test it at the real system.

5. Development tool

The software used to develop the Emulations is LabVIEW, from National Instruments. This development tool offers important advantages in comparison to similar simulation software, such as SIMIT, from Siemens, DELMIA, from Dassault Systems, COSIMIR, from Festo, and SPS-VISU, from MHJ-Software, among others.

Whereas some are very complex and expensive tools, others are cheaper but not so flexible. None of them allow a massive distribution of the developed models, since a license needs to be purchased; even if it is only desired to use already developed Emulations.

The main advantages of our development using LabVIEW to emulate processes are:

- **Distribution.** Once an Emulation has been developed, it can be distributed in a limitless way and completely free of charge to the students. The students just need to download the LabVIEW-Runtime from the Internet, which can be installed without the need of a license.
- **Portability.** Emulations can be used in computers running different operating systems. The development tool runs in diverse platforms such as: Windows, Mac OS, Solaris and Linux.
- **ActiveX Support.** LabVIEW supports the ActiveX functions. Although the Emulations were developed to be used with Siemens software, they can be easily adapted to work with software from different companies that support the ActiveX Technology.
- **OPC Support.** LabVIEW also supports the OPC Technology, which offers an open protocol to exchange data between applications from different companies. In the case of the Emulations, the OPC Technology can be used to control the Emulations using *real* PLC's from different providers. When the Emulations are controlled by *real* PLC's, the students can build up and then validate the required hardware configuration of the PLC, complementing the entire process.

6. Developed libraries

The connection between the Emulations and the simulated or the real PLC is achieved with a set of self developed libraries, which makes use of LabVIEW's ActiveX Technology support. Depending on the desired connection (whether with a simulated or with a real PLC), an appropriate ActiveX Control is used.

Besides, when working with real Siemens Controllers, the libraries allow the connection through almost all the PLC supported industrial networks, such as MPI, PROFIBUS, and Industrial Ethernet.

The libraries have been divided in 5 groups accordingly to the LabVIEW's programming standard: configuration, open, write, read, and close section. Furthermore, the error codes have been analyzed and prepared in such a way that, if a problem occurs, the user can easily find it out.

The libraries also take advantage of the polymorphic property, in order to write or read different data types with the same write or read function.

Figure 7 shows a typical connection with PLCSim. One can observe the way to write and to read an entire input and output byte, respectively.

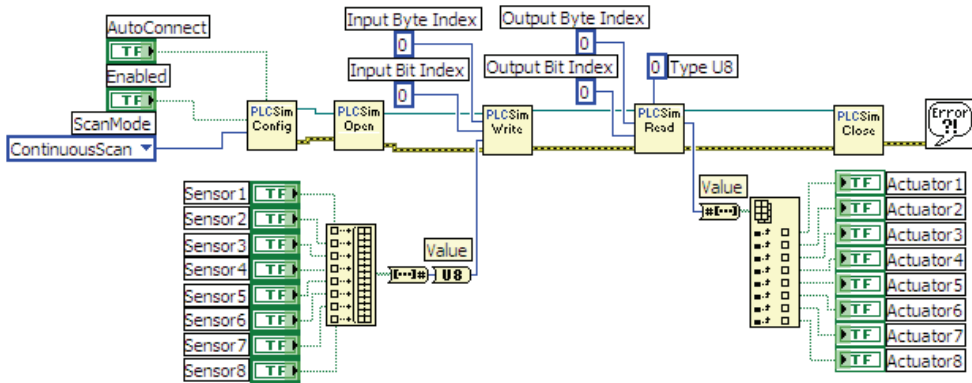


Fig. 7. Typical PLCSim connection

It is also possible to create new libraries that work with PLC environments from different providers, as long as the ActiveX technology is supported. It would be enough to select the adequate ActiveX Control and continue the scheme followed until now.

7. Student's perception

A survey was applied to all the students currently working with the Emulations, with the intention of knowing their perception about them. The students could rank each affirmation in a scale from 1 to 5; being 5 the highest score (Excellent) and 1 the worst score (Bad).

The obtained results (see Figure 8) show that a 77 % of the students think that the Emulations help them in an excellent way (score 5) to solve all of their assignments, whereas the remaining 23 % say that they did it in a very good way (score 4). An 84 % of the students believe that the Emulations support the learning process at the laboratory in an excellent way (score 5), whereas the remaining 16% think that they did it in a very good way (score 4). An 84 % of the students believe that the Emulations helped them to understand the concepts taught in class in an excellent way (score 5), whereas the remaining 16% think that they did it in a very good way (score 4).

Finally, a 65% of the students believe that, in general, the use of the Emulations on Automation Laboratories is Excellent, whereas the remaining 35 % think that it is Very Good.

It is important to emphasize that all of the results from the survey presented an Excellent (5) and Very Good (4) score; none of them received a score of 3 or less points.

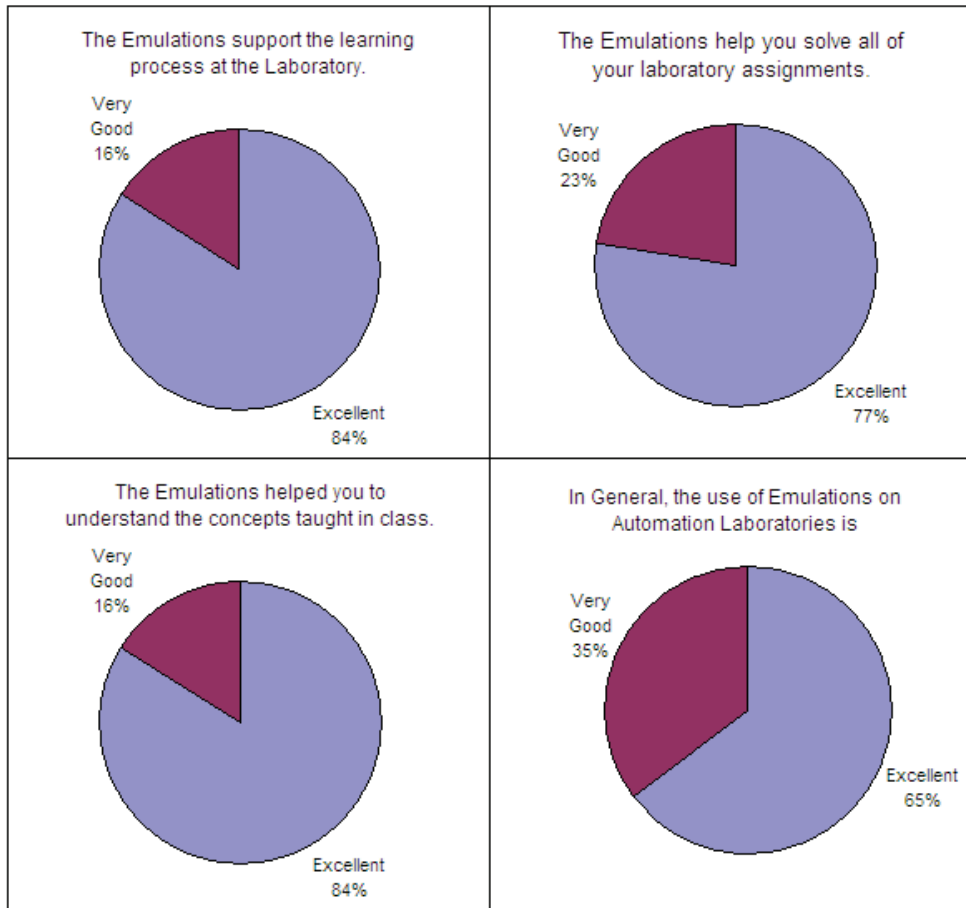


Fig. 8. Survey Results

8. Conclusions

Because of the high costs for appropriate laboratory equipment, one possible solution is to use computer emulations to support practical education and training.

In comparison with similar existing simulation software on the market, the referred computer emulations can be limitlessly distributed without restrictions and *without having to purchase any license*.

By handing the emulations to the students, *the laboratory concept is extended*. The students are then able to program, test, and debug their PLC programs without being restricted to a laboratory session. The laboratory is therefore always available.

The results obtained have shown that the students perform much better than in a traditional laboratory session. This fact has been confirmed after 3 years of using the Emulations as a part of the Tele-Engineering Lab at Monterrey Tech. The process has shown that the most important benefits are:

- Students solve the exercises in a more enthusiastic way, which influences directly on their performance.
- Students accepted working with Emulations and they are even pleased with its use, because of the fact that they offer them more flexibility when solving their assignments.
- Students enhance their learning process and as a result they also increase their final grades.
- Instructors have said that it is much easier to review all of the exercises, since they can make different tests to a student's program.

9. References

- [1] Fernández, Vicente, and Jiménez, Virtual Laboratories for Control Education: a Combined Methodology, *Int. J. Engng Ed.*, Vol. 21, No. 6, pp. 1059-1067, 2005
- [2] Liu, J., and Landers, R. G., Modular Control Laboratory System with Integrated Simulation, Animation, Emulation and Experimental Components, *Int. J. Engng Ed.*, Vol. 21, No. 6, pp. 1005-1016, 2005
- [3] Szczerbicka, Banks, Rogers, and Zeigler, "Conceptions of Curriculum for Simulation Education (Panel)", *Proceedings of the 2000 Winter Simulation Conference*, 2000.
- [4] Größler, A., "Don't let history repeat itself--methodological issues concerning the use of Simulators in teaching and experimentation", *System Dynamics Review ABI/INFORM*, Vol. 20, No. 3, 2004, pp. 263.
- [5] Macías, M., Méndez, I., TeleLab - Remote Automations Lab in Real Time, *38th ASEE/IEEE Frontiers in Education Conference*, 2008
- [6] Kezunovic, M., Abur, A., Huang, G., Bose A. and Tomsovic, K., "The Role of Digital Modeling and Simulation in Power Engineering Education", *IEEE transactions on power systems*, Vol. 19, No. 1, 2004 pp. 64-72.
- [7] Luo, W., Stravers, J., A., Duffin, K., L., "Lessons Learned from Using a Web-based Interactive Landform Simulation Model (WILSIM) in a General Education Physical Geography Course", *Journal of Geoscience Education, Bellingham*, Vol.53, No. 5, 2005, pp. 489.
- [8] Debevec, K., Shih, Kashyap, V., "Learning Strategies and Performance in a Technology Integrated Classroom", *Journal of Research on Technology in Education*, Vol.38, No. 3, 2006, pp. 293.
- [9] Macías, Méndez, Guridi, and Ortiz, TeleLab, Remote Laboratory for Automation and Control, *IFAC Conference on Cost Effective Automation IFAC-CEA 2007*.
- [10] Gomis, Montesinos, Galceran, Bergas and Sudriá, "A Chemical Process Automation Virtual Laboratory to Teach PLC Programming", *Int. J. Engng Ed.*, Vol. 23, No. 2, pp. 403-410, 2007
- [11] Rodríguez, Berenguel, and Guzmán, "A Virtual Course on Automation of Agriculture Systems", *Int. J. Engng Ed.*, Vol. 22, No. 6, pp. 1197-4101210, 2006

PLC based Structure for Management and Control of Distributed Energy Production Units

Joao M. G. Figueiredo
*CEM-IDMEC, Universidade Évora, Mechatronics Group
R. Romão Ramalho, 59; 7000-671 Évora,
Portugal*

1. Introduction

Renewable Energy consists of energy generated from natural and unlimited sources, which include, among others, wind, solar, biomass and hydroelectricity. These energy sources, unlike the fossil fuels, do not contribute to the greenhouse gas emissions, namely the carbon dioxide emissions, and do not suffer from depletion as well.

The global environmental alertness to protect Earth from the devastation of global warming has widespread, and consequently, governments' incentive policies have driven to a massive investment in renewable energy, namely wind power. These incentives not only avoid a direct competition between clean energy and the one obtained from the conventional technologies, but also minor the damages on environment and mitigate human activity impacts on ecosystems. The continuous growth of the world demand has conducted to an increase of the total electricity power installed capacity, including renewable energy.

The European electricity grid is one of the largest power systems in the world. Due to economies of scale it has always been advantageous to increase its size. Most of the European countries are synchronously connected to his grid. This means that the frequency in all of these interconnected countries is identical (in steady-state). The vast majority of the electricity in this grid is produced with large synchronous generators. Due to environmental, economical and geopolitical reasons there will be a shift in the production of energy. More and more distributed generation will be integrated in the system, some of which have significantly different characteristics when compared to the existing large synchronous generators (Doherty et al., 2005).

There should always be a balance between the supply and the demand of electricity. Any deviation results in a change of the frequency of 50Hz. A set of ancillary services is in use to control the frequency, and therefore the power balance of the grid.

In order to confront the variable or even stochastic behavior of the Renewable Energy Sources (RES), usually not meeting the electricity grid's demand, the adaptation of an appropriate Energy Storage System (ESS) is thought to be essential. On the other hand, storage techniques are faced with controversies mainly referring to the high initial cost rates, the additional transformation losses and the noteworthy environmental impacts, largely depending on the correlation between the type of technology used and the selected site (Denholm & Kulcinski, 2004).

Additionally, the common instability of electrical grids and the requirement for complete control over the quality of the electrical energy provision (Papathanassiou & Boulaxis, 2005), (Lund & Munster, 2003), set some serious obstacles in the dynamic exploitation of RES in autonomous electrical networks, this leading to the introduction of an upper limit of instantaneous RES contribution equal to a pre-described percentage (e.g. 30%) of the corresponding electricity demand.

To confront the problem described, several authors have every so often proposed alternative supply concepts such as water-pumping solutions, hydrogen storage, battery schemes and hybrid systems (Kaldellis, 2007), (Kaiser, 2007).

The system studied in this paper, illustrated in Fig. 1, supports the prior exploitation of RES in collaboration with state of art internal combustion engines set to operate in the range of minimum specific fuel consumption (maximum efficiency), while the ESS adopted is used to meet the satisfaction of power quality issues. In case of energy surplus, the excess amounts of energy are used to charge the ESS. When increased load demand and low RES production rates appear, the energy content of ESS is used and, if necessary, the programmed control of the thermal power stations calls for the back-up engines to set on.

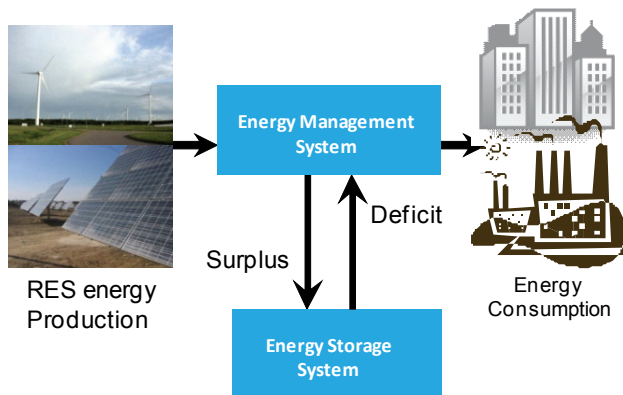


Fig. 1. RES production system integrated with ESS units

Today, the improvements in system communications have stimulated the implementation of distributed systems. These distributed systems are then usually managed by a centralized supervisory platform, commonly known as a SCADA system (Supervisory Control And Data Acquisition). This strategy reaches different fields, from agriculture, to industry, building automation, etc (Figueiredo & Botto, 2005), (Figueiredo & Sá Costa, 2007). An optimal-performance supervisory system has the objective to allocate the minimal needed power generation to the traditional power plant in order to produce the electricity at a minimal economic cost.

This paper presents a supervisory system to monitor and control energy production and consumption, in an optimized way. The developed system consists of a network of Programmable Logic Controllers (PLC), controlling locally the electricity production in each source, and measuring, in a real time base, the power consumption and production. The PLC network is parameterized according to the traditional Master-Slave requirements, using the PROFIBUS communication (Siemens, 2001). A SCADA system is implemented in order to supervise the entire PLC network.

This monitoring and control strategy is simulated based on the requirements of the renewable energy park that is being assembled in Évora University. This experimental park is founded by an European project (PETER) with Évora University - Portugal and Extremadura University - Spain. The PETER park is a renewable energy park that plans to include a photovoltaic unit (10 KW), a wind generator (1KW) and a biomass unit (75KW).

2. System model

The power plant studied in this paper is composed by several production units, spatially distributed, with different energetic sources: RES (Photovoltaic, Wind, Biomass), and oil-based thermal power stations for back-up purposes. Additionally this system contains also ESS systems. The electrical schematics of the developed multiple power-source system, with an ESS (battery) is illustrated in Fig. 2.

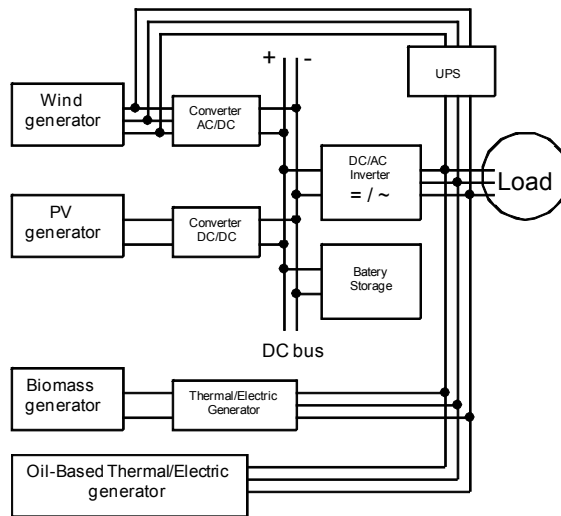


Fig. 2. Electrical schematics of multiple power generation

This paper develops a methodology that can be used with anyone of the several ESS available in the actual market. In fact, the choice of the proper ESS to fit to a specific application depends mainly on the storage requirements. Table 1 shows the most usual ESS systems and their applicability range dependent on load demand (Kaldellis et al., 2007).

ESS Type	Power Supply range
Flywheels	ca. 100KW
Li-ion batteries	100KW to 1MW
Lead-acid batteries	100KW to 10MW
Na-S batteries	100KW to 10MW
Fuel cells	100KW to 10MW
Flow batteries	100KW to 10MW
Pumped hydro	1MW to 100MW
CAES	1MW to 100MW

Table 1. Common Energy Storage Systems (ESS)

In the specific application simulated in this paper (PETER Park) the ESS Pumped Hydro received a particular emphasis as this park is located in an agriculture land with strong irrigation requirements. The usual developed technology used for the pumped hydro is illustrated in figure 3.

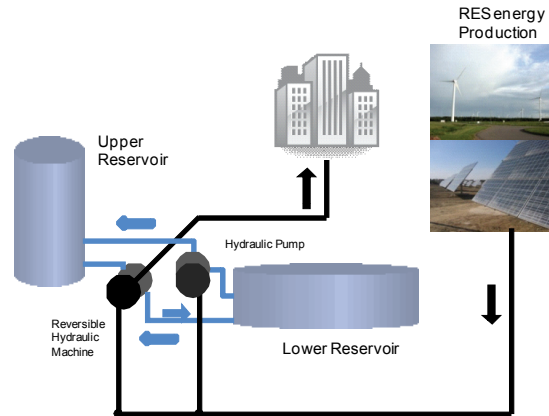


Fig. 3. ESS - Pumped Hydro

Each power source is connected to the central management platform through a typical PLC master-slave network. The master PLC communicates with a SCADA system that enlarges the system communication capabilities, allowing on-line monitoring and control, events recording, alarm management, etc.

In each power unit there exists a slave PLC, which is connected to the master PLC through a Profibus network. This power unit PLC monitors and controls the on-line power delivery to the electric grid. Similarly each ESS has a slave PLC controlling the income/outcome energy in the system.

3. Control strategy

The developed strategy is implemented through a traditional cascade controller. The inner-loop control is performed by a PLC network controlling locally each power plant. The outer-loop is managed by a SCADA supervisory system.

Each PLC hosts several control programs whose selection is made either locally, via an HMI (Human Machine Interface) or remotely, via the Master PLC. The Master PLC is connected to the server PC, via RS232/ MPI Siemens protocol, where the SCADA application is running.

The server PC is simultaneous a SCADA server and an internet server, as the implemented SCADA application is web enabled. All process variables are available at the SCADA PC as these variables are on-line available through a Profibus/ DP connection protocol (Siemens, 2001a).

3.1 SCADA outer-loop controller

A Supervisory Control and Data Acquisition (SCADA) System is used as an application development tool that enables system integrators to create sophisticated supervisory and control applications for a variety of technological domains, mainly in the industry field. The

main feature of a SCADA system is its ability to communicate with control equipment in the field, through the PLC network. As the equipment is monitored and data is recorded, a SCADA application responds according to system logic requirements or operator requests. In the developed control strategy, the SCADA application performs the outer control loop of the energy plant system. At this outer loop several complex control structures can be used to manage the overall system dynamics.

In this paper an optimal allocation of production resources is performed taking into account the minimization of the operational costs, what usually corresponds, in an hybrid power system, to the minimization of the supplied power from the oil-based thermal stations. Both instantaneous power demand and power supply are on-line monitored in the developed energy management system.

Considering the application developed in this paper (isolated production/consumption system) the on-line monitoring of the power demand is performed by reading the power delivered at the output of the main electric panel.

In order to guarantee the stability and quality of the electric power delivered, a set of Energy Storage Systems and back-up oil-based thermal power stations are integrated in the production system.

Assuming that the projected hybrid power plant had been optimal designed (Shaahid & Elhadidy, 2008), the role of the platform here developed is basically to minimize the energy supplied by the oil-based back-up power units. We use the potential of the SCADA supervisory platform to integrate the monitoring of the real production figures on the optimization problem.

The selected functional to allocate the proper electricity production to each power unit is presented below (eqs. 1 to 7):

$$\min J = \sum_i c_{PV_i} y_{PV_i} + \sum_j c_{wind_j} W_{wind_j} + \sum_l c_{oil_l} W_{oil_l} \tag{1}$$

Subjected to:

$$\sum_i y_{PV_i} + \sum_j y_{wind_j} + \sum_l y_{oil_l} + \sum_n y_{ESS_n} \geq y_{demand} \tag{2}$$

$$y_{ESS_k} \leq y_{max_k} \tag{3}$$

$$E_{max_k} \geq y_{ESS_k} \times \Delta t_k \tag{4}$$

$$0 \leq y_{PV_k} \leq y_{PV_{kact}} \tag{5}$$

$$0 \leq y_{wind_k} \leq y_{wind_{kact}} \tag{6}$$

$$0 \leq y_{oil_k} \tag{7}$$

where:

c_{PV_i} = production cost associated with PV plant i ;

c_{wind_i} = production cost associated with Wind plant i ;

c_{oil_i} = production cost associated with oil-based thermal plant i ;

y_{PVi} = requested Watt-power to be supplied by PV plant i ;
 y_{windi} = requested Watt-power to be supplied by wind plant i ;
 y_{oili} = requested Watt-power to be supplied by oil-based thermal plant i ;
 y_{ESSi} = requested Watt-power to be supplied by energy storage system i ;
 y_{demand} = total Watt-power demand;
 y_{maxk} = max available Watt-power to be supplied by energy storage system k ;
 E_{maxk} = max available Joule-energy to be supplied at an average rate of y_{ESS} , by a time periode of Δt , for the ESS k ;
 y_{PVkact} = instantaneous available Watt-power at the PV plant k ;
 $y_{windkact}$ = instantaneous available Watt-power at the wind plant k ;

Analysing the minimization criterion, it is clear that the change at the instantaneous available watt-power from the RES (y_{PVkact} , $y_{windkact}$) implies the energy re-balance of the entire system. In fact when it happens a power surplus (production greater than consumption), the Energy Storage Systems are being charged ($y_{ESSk} < 0$). When the demanded power exceeds the RES production, the difference has to be covered by additional requirements on ESS supply ($y_{ESSk} > 0$) or oil-based thermal systems production ($y_{oilk} > 0$).

The optimization algorithm implemented for the energy management, at the SCADA outer loop control, could not be implemented directly on the SCADA system, as this complex controller needs mathematical operations that are not present at usual available SCADA systems. In this paper we developed a strategy to couple the SCADA system with the MATLAB software (Mathworks, 2005).

The communication between SCADA and MATLAB was performed using the DDE protocol (Dynamic Data Exchange). This communication protocol, developed in the 90's but still very common, permits the exchange of data between two independent running software programs (Client and Server).

In the developed application the MATLAB software was the Client, as it initiates the communication, and the SCADA software was the Server, as it responds to Client's requests. Among the different information formats supported by DDE protocol, the TEXT format was selected as this format was supported by both software: SCADA and MATLAB.

Figure 4 illustrates the communication flow that was developed to implement the optimization algorithm at the Outer loop control (eqs. 1 - 7). In this figure we see the coexistence of four different communication protocols (LAN, DDE, MPI, PROFIBUS) working simultaneously at different levels of the developed platform.

3.2 Local PLC inner-loop controller

At the inner loop of the developed strategy (PLC level), several algorithms had been developed. These algorithms were built using the Grafcet methodology - Sequential Function Chart. The designed algorithms were implemented using the Ladder Diagram language (Siemens, 2001b).

The main purpose of the developed programs associated with the RES stations is the monitoring of the electric power generated.

4. Experimental setup

The developed application to monitor and control automatic Power Plants had been implemented on an experimental setup with the following software and hardware requirements.

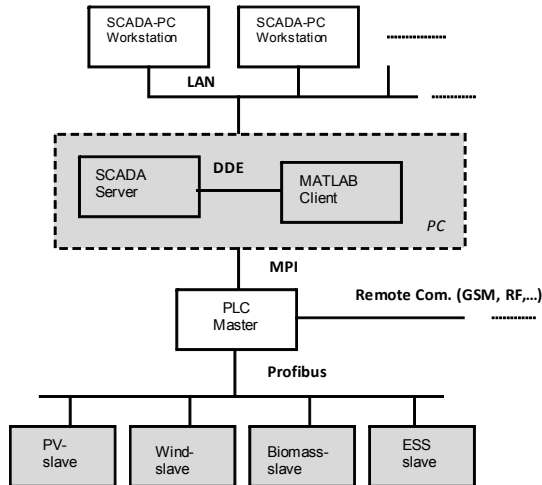


Fig. 4. Communication's Architecture for the built Prototype

The PLC network implemented had four PLCs: one PLC for each Power unit (PV-slave, Wind-slave, Biomass-slave, ESS-slave). Figure 4 shows the architecture of the built Prototype. This prototype aims to test the developed energy management system to be implemented in the future Peter park.

The PETER park plans to integrate a photovoltaic production unit (10 KW), a wind generator (1KW) and a biomass unit (75KW). In this park, the biomass plant will play the role of the controllable power production unit.

4.1 Software requirements

The software used for the PLC programming was the Siemens Simatic Step 7 (Siemens, 2000). The Scada system was developed over the platform Siemens WinCC (Siemens, 2005).

4.2 Hardware requirements

Figure 5 shows an overview of the implemented prototype. Referring hardware characteristics each PLC (Master and Slaves) was composed by the following Siemens modules:

Slot1 = Power supply PS 307-2A

Slot2 = Processor CPU 315-2DP

Slot4 = Communication module CP 342 -5

Slot5 = Digital card DI8/DO8xDC24V/0,5A

Slot6 = Digital card DI8/DO8xDC24V/0,5A

Slot7 = Analogue card AI4/ AO2x8/ 8bit

Additionally, the Master PLC has a modem for GSM communication that provides the system capacity to communicate through the mobile phone network.

The sensors used to monitor the generated and consumed electric power/ current are a set of AC/DC current transducers, coupled to energy analysers, with Profibus communication. In our case the energy meters used were the family Siemens SIMEAS P.

The power generation of the considered RES units, was simulated through 2 DC-power supplies, and 1 AC-Power supply, which simulated the power output from the DC-converters and the AC-generator illustrated in fig. 2. The power amplitude was externally changed.

5. Results

The main objective of the performed tests was to evaluate the compatibility of the several communication protocols present in the developed application (LAN, DDE, MPI, PROFIBUS).

The obtained results show mainly the information made available at the several developed Graphical User Interfaces (GUI) of the application.

The optimization problem described in eqs. 1 to 7, was solved through the MATLAB Optimization Toolbox, using the standard algorithm "fmincon" (Mathworks, 2005).

The SCADA system used to implement this monitoring and control strategy permits the selective access to the application, depending on the user's responsibility degree. In this paper we developed three user levels: Operators, Supervisors and Administrators.

Several SCADA menus were built. The main characteristic of a SCADA Menu is to be simple, explicit and quick on transmitting the information to the operator or to the System Administrator.

Two of the developed Graphical User Interfaces (GUI) are shown in figs 6 and 7.

As this SCADA platform is web enabled, all the GUI displayed data is also on-line accessible through the internet.

In fig. 6 it is shown an overview of the complete Power Plant production, with the main information regarding the consumption and the production on several distributed power units (PV, Wind, Biomass). The on-line available information, referring actual data from each power unit is: actual values and maximal daily values for Voltage, Current, Power and efficiency ratio (actual Value/max. Value). In fig. 7 it is shown the GUI relative to one of the power units. In this case this figure shows the available information for the PV-power generator, concerning the several integrated sub-systems.

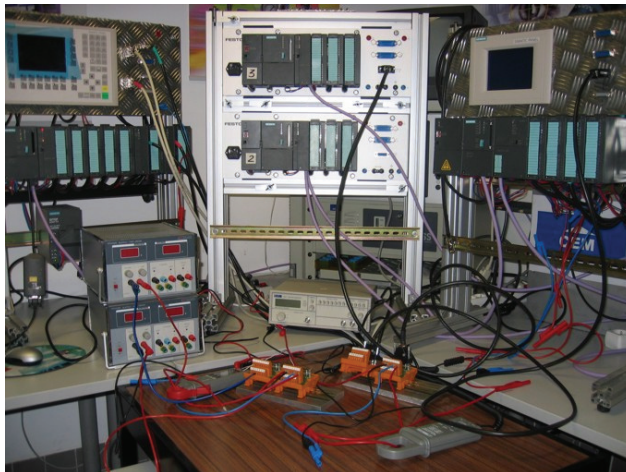


Fig. 5. Implemented Prototype to test the energy management system

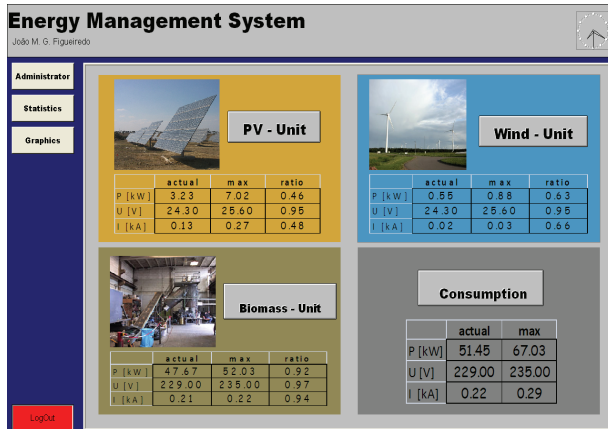


Fig. 6. GUI: overview of the complete Power Plant Production

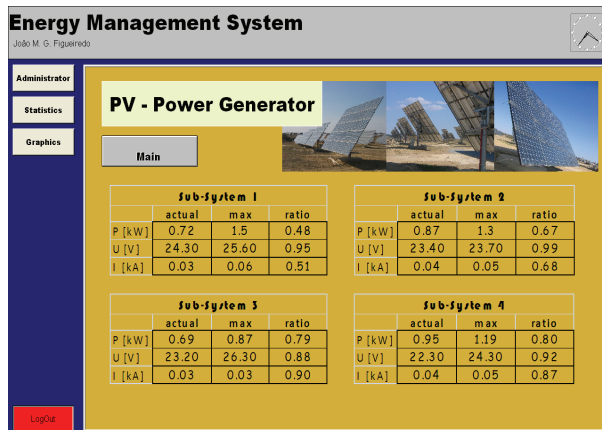


Fig. 7. GUI: Photovoltaic Power Generation

6. Conclusion

The energy management system developed in this paper is composed by several production units, spatially distributed, with different energetic sources: Renewable Energy Sources - RES (Photovoltaic, Wind, Biomass), Oil-based thermal power stations and Energy Storage Systems - ESS.

The developed strategy is implemented through a traditional cascade controller.

The inner-loop control is performed by an industrial PLC network, controlling locally each power plant. The outer-loop is managed by a SCADA supervisory system.

In this paper an optimal allocation of production resources is performed taking into account the minimization of the operational costs. Both instantaneous power-demand and power-supply are on-line monitored in the developed energy management system.

The developed strategy is simulated, based on the requirements of the new renewable energy experimental park (PETER), that is being implemented at the University of Évora - Portugal.

7. References

- Denholm P., Kulcinski, G. (2004). Life Cycle Energy Requirements and Greenhouse Gas Emissions from Large Scale Energy Storage Systems. *Energy Conversion Manag.* 2004, 45(13-14), pp. 2153-2172.
- Doherty, R., Lalor, G., O'Malley, M. (2005) "Frequency control in competitive electricity market dispatch"; *IEEE Trans. on Power Systems*, vol 20, n°3, pp. 1588-1596
- Figueiredo, J., M. Botto (2005). Automatic Control Strategies Implemented on a Water Canal Prototype. *Proc. 16th IFAC World Congress, Praha, Czec Republic*
- Figueiredo, J., Sá da Costa J. (2007). A Concept for an Operational Management System for Industrial Purposes. *Proc. IEEE Intl. Symposium on Intelligent Signal Processing, Madrid, Spain, ISBN 1-4244-0830-X/07, 2007 IEEE*
- Kaiser, R. (2007). Optimized Battery-Management System to Improve Storage Lifetime in Renewable Energy Systems. *J. Power Systems 2007*, 168, pp. 58-65.
- Kaldellis, J. (2007). An Integrated Model for Performance Simulation of Hybrid Wind-Diesel Systems. *Energy 32 (2007)*, pp. 1544-1564
- Lund, H., Munster, E. (2003). Management of Surplus Electricity - Production from a Fluctuating Renewable-Energy Source. *Appl. Energy 2003*, 76(1-3), pp. 65-74.
- Mathworks (2005). *Matlab Simulink 7.1 (R14)*, Mathworks, 2005.
- Papathanassiou St A., Boulaxis N. (2005). Power Limitations and Energy Yield Evaluation for Wind Farms Operating in Island Systems. *Renew Energy 2005*, 31(4), pp. 457-479.
- Shaahid, S., Elhadidy, M. (2008). Economic Analysis of Hybrid Photovoltaic-diesel-battery power systems for residential loads in hot regions - A step to clean future. *Renewable and Sustainable Energy Rev 12 (2008)*, pp. 488-503.
- Siemens (2000). *System Software for S7-300 and S7-400 - Reference Manual*, SIEMENS 08/2000; A5E00069892-02
- Siemens (2001a). *Simatic Net - NCM S7 for Profibus/FMS*. SIEMENS 12/2001.
- Siemens (2001b). *Simatic S7-300 - Ladder Logic (LAD) for S7-300*, SIEMENS, 2001.
- Siemens (2005). *Simatic WinCC V6.0 SP2*, SIEMENS, 2005.