IEC 62541-10

Edition 2.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC Unified Architecture – Part 10: Programs

Architecture unifiée OPC – Partie 10: Programmes

IEC 62541-10:2015-03(en-fr)

Copyright International Electrotechnical Commission

THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2015 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 15 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 60 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

Copyright International Electrotechnical Commission

IEC 62541-10

Edition 2.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC Unified Architecture – Part 10: Programs

Architecture unifiée OPC – Partie 10: Programmes

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100

ISBN 978-2-8322-2274-4

Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

® Registered trademark of the International Electrotechnical Commission

Copyright International Electrotechnical Commission, de la Commission Electrotechnique Internationale

CONTENTS

Copyright International Electrotechnical Commission

Copyright International Electrotechnical Commission

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62541-10 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2012. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) Based on NIST review, security considerations have been included as 4.2.2;
b) Fixed the definition of the Program Diagnostic Type into a data type (5.2.8) and added missing data type for the Program Diagnostic Variable in the ProgramType in Table 5.
c) Corrected the BrowseName of the audit events for Program Transitions in Table 7.

Copyright International Electrotechnical Commission

The text of this standard is based on the following documents:

FDIS	Report on voting
65E/383/FDIS	65E/409/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

Copyright International Electrotechnical Commission

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture (OPC UA) standard series and defines the information model associated with *Programs*. This includes the description of the *NodeClasses,* standard *Properties, Methods* and *Events* and associated behaviour and information for *Programs*.

The complete address space model including all *NodeClass*es and *Attributes* is specified in IEC 62541-3. The services such as those used to invoke the *Methods* used to manage P*rograms* are specified in IEC 62541-4.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

IEC 62541-3:2015, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4:2015, *OPC Unified Architecture – Part 4: Services*

IEC 62541-5:2015, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

3 Terms, definitions and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC 62541-3, as well as the following apply.

3.1.1
function
programmatic task performed by a server or device, usually accomplished by computer code execution

3.1.2
Finite State Machine
sequence of states and valid state transitions along with the causes and effects of those state transitions that define the actions of a *Program* in terms of discrete stages

3.1.3
ProgramType
type definition of a *Program* and is a subtype of the *FiniteStateMachineType*

Copyright International Electrotechnical Commission

3.1.4
Program Control Method
Method having specific semantics designed for the control of a *Program* by causing a state transition

3.1.5
Program Invocation
unique *Object* instance of a *Program* existing on a *Server*

Note 1 to entry: A *Program Invocation* is distinguished from other *Object* instances of the same *ProgramType* by the object node's unique browse path.

3.2 Abbreviations

DA Data Access
FSM Finite State Machine
HMI Human Machine Interfaces
PCM Program Control Method
PGM Program
PI Program Invocation
UA Unified Architecture

4 Concepts

4.1 General

Integrated automation facilities manage their operations through the exchange of data and the coordinated invocation of system functions as illustrated in Figure 1. *Services* are required to perform the data exchanges and to invoke the functions that constitute system operation. These functions may be invoked through Human Machine Interfaces, cell controllers, or other supervisory control and data acquisition type systems. OPC UA defines *Methods* and *Programs* as an interoperable way to advertise, discover, and request these functions. They provide a normalizing mechanism for the semantic description, invocation, and result reporting of these functions. Together *Methods* and *Programs* complement the other OPC UA *Services* and *ObjectTypes* to facilitate the operation of an automation environment using a client-server hierarchy.

Copyright International Electrotechnical Commission

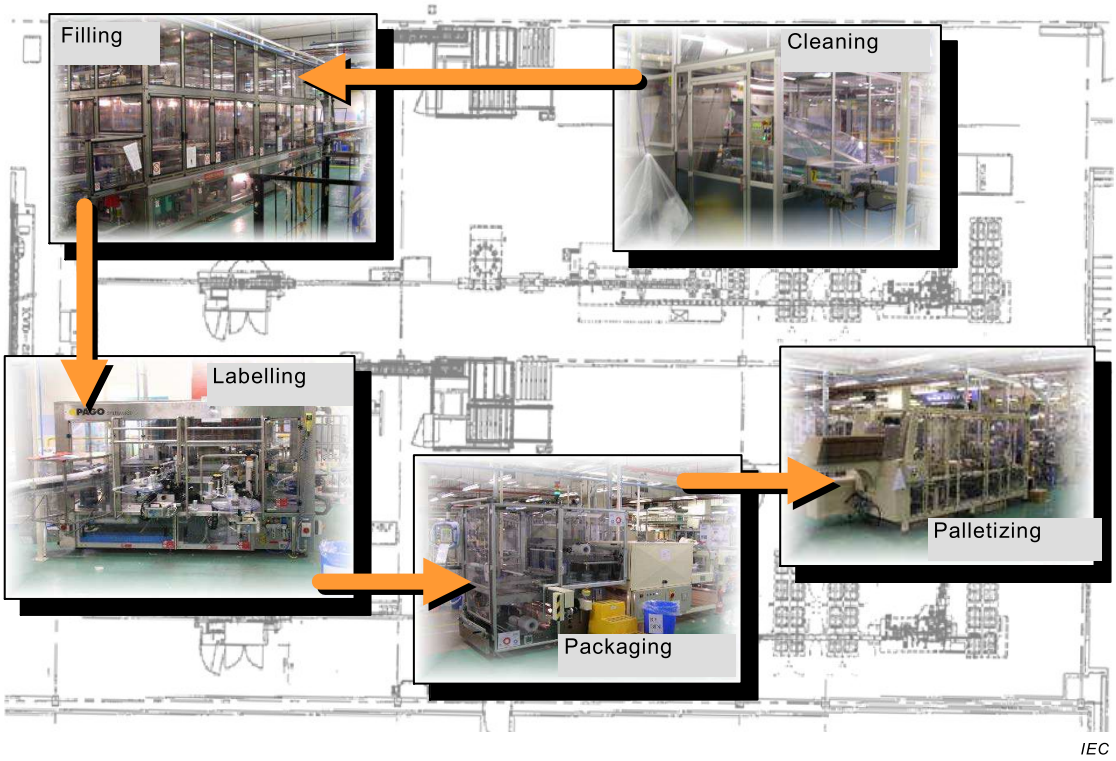

Figure 1 – Automation facility control

Methods and *Programs* model functions typically have different scopes, behaviours, lifetimes, and complexities in *OPC Servers* and the underlying systems. These functions are not normally characterized by the reading or writing of data which is accomplished with the OPC UA A*ttribute* service set.

Methods represent basic functions in the *Server* that can be invoked by a *Client*. *Programs,* by contrast, model more complex and stateful functionality in the system. For example, a method call may be used to perform a calculation or reset a counter. A *Program* is used to run and control a batch process, execute a machine tool part program, or manage a domain download. *Methods* and their invocation mechanism are described in IEC 62541-3 and IEC 62541-4.

This standard describes the extensions to, or specific use of, the core capabilities defined in IEC 62541-5 as required for *Programs*.

4.2 Programs

4.2.1 Overview

Programs are complex functions in a server or underlying system that can be invoked and managed by a *Client*. *Programs* can represent any level of functionality within a system or process in which client control or intervention is required and progress monitoring is desired. Figure 2 illustrates the model.

Copyright International Electrotechnical Commission

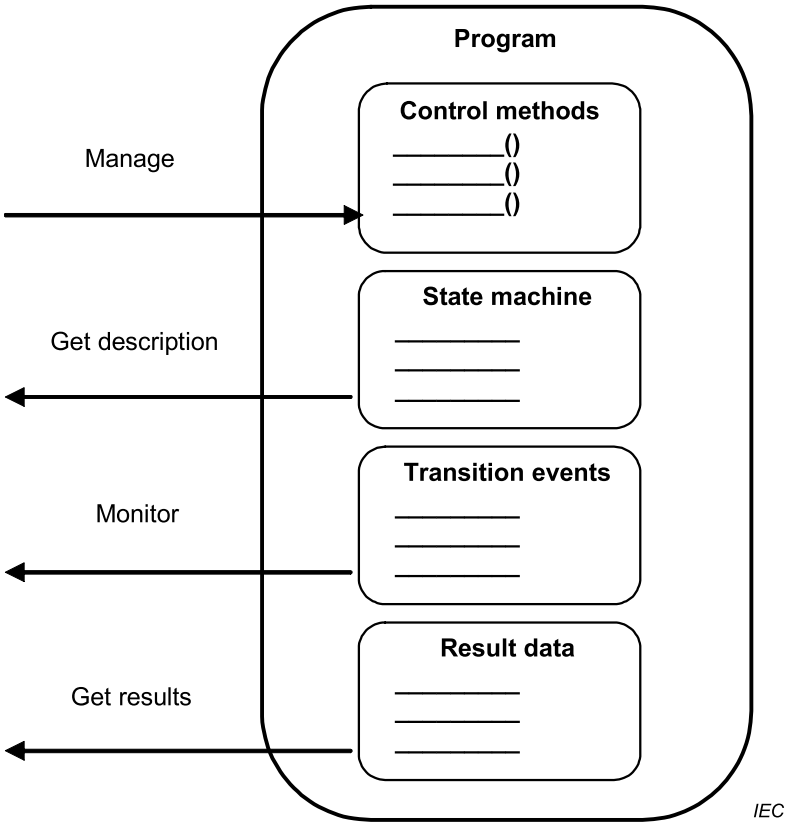

Figure 2 – Program illustration

Programs are state full and transition through a prescribed sequence of states as they execute. Their behaviour is defined by a *Program Finite State Machine (PFSM)*. The elements of the PFSM describe the phases of a *Program's* execution in terms of valid transitions between a set of states, the stimuli or causes of those transitions, and the resultant effects of the transitions.

4.2.2 Security considerations

Since *Programs* can be used to perform advanced control algorithms or other actions, their use should be restricted to personnel with appropriate access rights. It is recommended that *AuditUpdateMethodEvents* are generated to allow monitoring the number of running *Programs* in addition to their execution frequency.

4.2.3 Program Finite State Machine

The states, transitions, causes and effects that compose the *Program Finite State Machine* are listed in Table 1 and illustrated in Figure 3.

Table 1 – Program Finite State Machine

No.	Transition name	Cause	From state	To state	Effect
1	HaltedToReady	Reset Method	Halted	Ready	Report Transition 1 Event/Result
2	ReadyToRunning	Start Method	Ready	Running	Report Transition 2 Event/Result
3	RunningToHalted	Halt Method or Internal (Error)	Running	Halted	Report Transition 3 Event/Result
4	RunningToReady	Internal	Running	Ready	Report Transition 4 Event/Result
5	RunningToSuspended	Suspend Method	Running	Suspended	Report Transition 5

Copyright International Electrotechnical Commission

No.	Transition name	Cause	From state	To state	Effect
					Event/Result
6	SuspendedToRunning	Resume Method	Suspended	Running	Report Transition 6 Event/Result
7	SuspendedToHalted	Halt Method	Suspended	Halted	Report Transition 7 Event/Result
8	SuspendedToReady	Internal	Suspended	Ready	Report Transition 8 Event/Result
9	ReadyToHalted	Halt Method	Ready	Halted	Report Transition 9 Event/Result

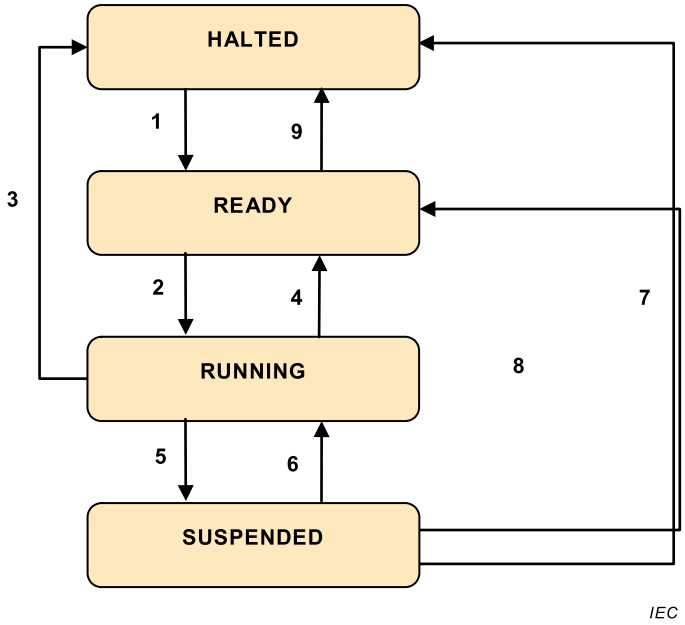

Figure 3 – Program states and transitions

4.2.4 Program states

A standard set of base states is defined for *Programs* as part of the *Program Finite State Machine.* These states represent the stages in which a *Program* can exist at an instance in time as viewed by a *Client*. This state is the *Program's* current state. All *Programs* shall support this base set. A *Program* may or may not require a *Client* action to cause the state to change. The states are formally defined in Table 2.

Table 2 – Program states

State	Description
Ready	The *Program* is properly initialized and may be started.
Running	The *Program* is executing making progress towards completion.
Suspended	The *Program* has been stopped prior to reaching a terminal state but may be resumed.
Halted	The *Program* is in a terminal or failed state, and it cannot be started or resumed without being reset.

The set of states defined to describe a *Program* can be expanded. *Program* sub states can be defined for the base states to provide more resolution of a process and to describe the cause and effect(s) of additional stimuli and transitions. Standards bodies and industry groups may extend the base *Program Finite State Model* to conform to various industry models. For example, the Halted state can include the sub states "Aborted" and "Completed" to indicate if

Copyright International Electrotechnical Commission

the function achieved a successful conclusion prior to the transition to Halted. Transitional states such as “Starting” or “Suspending” might also be extensions of the Running state, for example.

4.2.5 State transitions

A standard set of state transitions is defined for the *Program Finite State Machine.* These transitions define the valid changes to the *Program’s* current state in terms of an initial state and a resultant state. The transitions are formally defined in Table 3.

Table 3 – Program state transitions

Transition no.	Transition name	Initial state	Resultant state
1	HaltedToReady	Halted	Ready
2	ReadyToRunning	Ready	Running
3	RunningToHalted	Running	Halted
4	RunningToReady	Running	Ready
5	RunningToSuspended	Running	Suspended
6	SuspendedToRunning	Suspended	Running
7	SuspendedToHalted	Suspended	Halted
8	SuspendedToReady	Suspended	Ready
9	ReadyToHalted	Ready	Halted

4.2.6 Program state transition stimuli

The stimuli or causes for a *Program’s* state transitions can be internal to the *Server* or external. The completion of machining steps, the detection of an alarm condition, or the transmission of a data packet are examples of internal stimuli. *Methods* are an example of external stimuli. Standard *Methods* are defined which act as stimuli for the control of a *Program*.

4.2.7 Program Control Methods

Clients manage a *Program* by calling *Methods.* The *Methods* impact a *Program’s* behaviour by causing specified state transitions. The state transitions dictate the actions performed by the *Program*. This standard defines a set of standard *Program Control Methods*. These *Methods* provide sufficient means for a client to run a *Program*.

Table 4 lists the set of defined *Program Control Methods.* Each *Method* causes transitions from specified states and shall be called when the *Program* is in one of those states.

Individual *Programs* can optionally support any subset of the *Program Control Methods.* For example, some *Programs* may not be permitted to suspend and so would not provide the *Suspend* and *Resume Methods.*

Programs can support additional user defined *Methods.* User defined *Methods* shall not change the behaviour of the base *Program Finite State Machine*.

Copyright International Electrotechnical Commission

Table 4 – Program Control Methods

Method Name	Description
Start	Causes the *Program* to transition from the Ready state to the Running state.
Suspend	Causes the *Program* to transition from the Running state to the Suspended state.
Resume	Causes the *Program* to transition from the Suspended state to the Running state.
Halt	Causes the *Program* to transition from the Ready, Running or Suspended state to the Halted state.
Reset	Causes the *Program* to transition from the Halted state to the Ready state.

Program Control Methods can include arguments that are used by the *Program.* For example, a Start *Method* may include an options argument that specifies dynamic options used to determine some program behaviour. The arguments can differ on each *ProgramType*. The *Method Call* service specified in IEC 62541-4:2015, 5.11 defines a return status. This return status indicates the success of the *Program Control Method* or a reason for its failure.

4.2.8 Program state transition effects

A *Program's* state transition generally has a cause and also yields an effect. The effect is a by product of a *Program* state transition that can be used by a *Client* to monitor the progress of the *Program*. Effects can be internal or external. An external effect of a state transition is the generation of an *Event* notification. Each *Program* state transition is associated with a unique *Event*. These *Events* reflect the progression and trajectory of the *Program* through its set of defined states. The internal effects of a state transition can be the performance of some programmatic action such as the generation of data.

4.2.9 Program result data

4.2.9.1 Overview

Result data is generated by a running *Program.* The result data can be intermediate or final. Result data may be associated with specific *Program* state transitions.

4.2.9.2 Intermediate result data

Intermediate result data is transient and is generated by the *Program* in conjunction with non-terminal state transitions. The data items that compose the intermediate results are defined in association with specific *Program* state transitions. Their values are relevant only at the transition level.

Each *Program* state transition can be associated with different result data items. Alternately, a set of transitions can share a result data item. Percentage complete is an example of intermediate result data. The value of percentage complete is produced when the state transition occurs and is available to the *Client*.

Clients acquire intermediate result data by subscribing to *Program* state transition *Events*. The *Events* specify the data items for each transition. When the transition occurs, the generated *Event* conveys the result data values captured to the subscribed *Clients*. If no *Client* is monitoring the *Program*, intermediate result data may be discarded.

4.2.9.3 Terminal result data

Terminal result data is the final data generated by the *Program* as it ceases execution. Total execution time, number of widgets produced, and fault condition encountered are examples of terminal result data. When the *Program* enters the terminal state, this result data can be conveyed to the client by the transition *Event*. Terminal result data is also available within the *Program* to be read by a *Client* after the program stops. This data persists until the *Program* Instance is rerun or deleted.

Copyright International Electrotechnical Commission

4.2.9.4 Monitoring Programs

Clients can monitor the activities associated with a *Program's* execution. These activities include the invocation of the management *Methods*, the generation of result data, and the progression of the *Program* through its states. *Audit Events* are provided for *Method Calls* and state transitions. These *Events* allow a record to be maintained of the *Clients* that interacted with any *Program* and the *Program* state transitions that resulted from that interaction.

4.2.10 Program lifetime

4.2.10.1 Overview

Programs can have different lifetimes. Some *Programs* may always be present on a *Server* while others are created and removed. Creation and removal can be controlled by a *Client* or may be restricted to local means.

A *Program* can be *Client* creatable. If a *Program* is *Client* creatable, then the *Client* can add the *Program* to the *Server*. The *Object Create Method* defined in IEC 62541-3:2015, 5.5.4, is used to create the *Program* instance. The initial state of the *Program* can be Halted or Ready. Some *Programs*, for example, may require that a resource becomes available after its creation and before it is ready to run. In this case, it would be initialized in the Halted state and transition to Ready when the resource is delivered.

A *Program* can be *Client* removable. If the *Program* is *Client* removable, then the *Client* can delete the *Program* instance from the *Server*. The *DeleteNodes Service* defined in IEC 62541-4 is used to remove the *Program* Instance. The *Program* shall be in a Halted state to be removed. A *Program* may also be auto removable. An auto removable *Program* deletes itself when execution has terminated.

4.2.10.2 Program instances

Programs can be multiple instanced or single instanced. A *Server* can support multiple instances of a *Program* if these *Program* Instances can be run in parallel. For example, the *Program* may define a *Start Method* that has an input argument to specify which resource is acted upon by its functions. Each instance of the *Program* is then started designating use of different resources. The *Client* can discover all instances of a *Program* that are running on a *Server*. Each instance of a *Program* is uniquely identified on the *Server* and is managed independently by the *Client*.

4.2.10.3 Program recycling

Programs can be run once or run multiple times (recycled). A *Program* that is run once will remain in the Halted state indefinitely once it has run. The normal course of action would be to delete it following the inspection of its terminal results.

Recyclable *Programs* may have a limited or unlimited cycle count. These *Programs* may require a reset step to transition from the Halted state to the Ready state. This allows for replenishing resources or reinitializing parameters prior to restarting the *Program*. The *Program Control Method* "*Reset"* triggers this state transition and any associated actions or effects.

5 Model

5.1 General

The *Program* model extends the *FiniteStateMachineType and* basic *ObjectType* models presented in IEC 62541-5. Each *Program* has a *Type Definition* that is the subtype of the *FiniteStateMachineType.* The *ProgramType* describes the *Finite State Machine* model supported by any *Program Invocation* of that type. The *ProgramType* also defines the property

Copyright International Electrotechnical Commission

set that characterizes specific aspects of that *Program's* behaviour such as lifetime and recycling as well as specifying the result data that is produced by the *Program*.

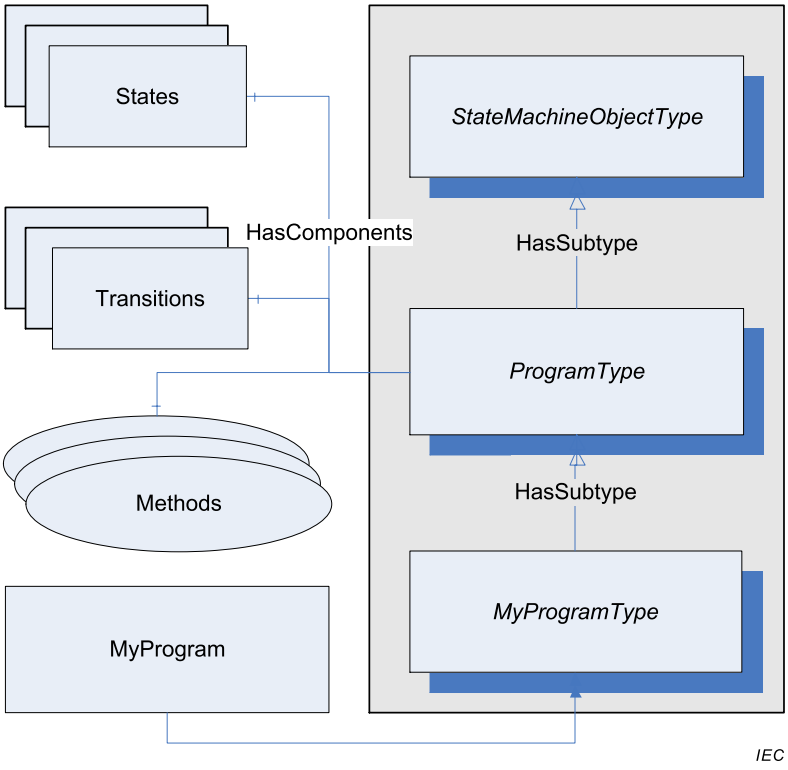

Figure 4 – Program Type

The base *ProgramType* defines the standard *Finite State Machine* specified for all *Programs*. This includes the states, transitions, and transition causes (*Methods*) and effects (*Events*). Subtypes of the base *ProgramType* can be defined to extend or more specifically characterize the behaviour of an individual *Program* as illustrated with "MyProgramType" in Figure 4.

5.2 ProgramType

5.2.1 Overview

The additional properties and components that compose the *ProgramType* are listed in Table 5. No *ProgramType* specific semantics are assigned to the other base *ObjectType* or *FiniteStateMachineType Attributes* or *Properties*.

Copyright International Electrotechnical Commission

Table 5 – ProgramType

Attribute	**Value**				
	Includes all attributes specified for the FiniteStateMachineType				
BrowseName	ProgramType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	Creatable	Boolean	PropertyType	--
HasProperty	Variable	Deletabe	Boolean	PropertyType	Mandatory
HasProperty	Variable	AutoDelete	Boolean	PropertyType	Mandatory
HasProperty	Variable	RecycleCount	Int32	PropertyType	Mandatory
HasProperty	Variable	InstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxInstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxRecycleCount	UInt32	PropertyType	--
HasComponent	Variable	ProgramDiagnostic	ProgramDiagnosticDataType	ProgramDiagnosticType	Optional
HasComponent	Object	Halted		StateType	Mandatory
HasComponent	Object	Ready		StateType	Mandatory
HasComponent	Object	Running		StateType	Mandatory
HasComponent	Object	Suspended		StateType	Mandatory
HasComponent	Object	HaltedToReady		TransitionType	Mandatory
HasComponent	Object	ReadyToRunning		TransitionType	Mandatory
HasComponent	Object	RunningToHalted		TransitionType	Mandatory
HasComponent	Object	RunningToReady		TransitionType	Mandatory
HasComponent	Object	RunningToSuspended		TransitionType	Mandatory
HasComponent	Object	SuspendedToRunning		TransitionType	Mandatory
HasComponent	Object	SuspendedToHalted		TransitionType	Mandatory
HasComponent	Object	SuspendedToReady		TransitionType	Mandatory
HasComponent	Object	ReadyToHalted		TransitionType	Mandatory
HasComponent	Method	Start			Optional
HasComponent	Method	Suspend			Optional
HasComponent	Method	Reset			Optional
HasComponent	Method	Halt			Optional
HasComponent	Method	Resume			Optional
HasComponent	Object	FinalResultData		BaseObjectType	Optional

Copyright International Electrotechnical Commission

5.2.2 ProgramType Properties

The *Creatable Property* is a boolean that specifies if *Program Invocations* of this *ProgramType* can be created by a *Client*. If False, these *Program Invocations* are persistent or may only be created by the *Server*.

The *Deletable Property* is a boolean that specifies if a *Program Invocation* of this *ProgramType* can be deleted by a *Client*. If False, these *Program Invocations* can only be deleted by the *Server*.

The *AutoDelete Property* is a boolean that specifies if *Program Invocations* of this *ProgramType* are removed by the *Server* when execution terminates. If False, these *Program Invocations* persist on the *Server* until they are deleted by the *Client*. When the *Program Invocation* is deleted, any result data associated with the instance is also removed.

The *RecycleCount Property* is an unsigned integer that specifies the number of times a *Program Invocation* of this type has been recycled or restarted from its starting point (not resumed). Note that the Reset Method may be required to prepare a *Program* to be restarted.

The *MaxRecycleCount Property* is an integer that specifies the maximum number of times a *Program Invocation* of this type can be recycled or restarted from its starting point (not resumed). If the value is less than 0, then there is no limit to the number of restarts. If the value is zero, then the Program may not be recycled or restarted.

The *InstanceCount Property* is an unsigned integer that specifies the number of *Program Invocations* of this type that currently exist.

The *MaxInstanceCount Property* is an integer that specifies the maximum number of *Program Invocations* of this type that can exist simultaneously on this *Server*. If the value is less than 0, then there is no limit.

5.2.3 ProgramType components

5.2.3.1 Overview

The *ProgramType* components consist of a set of *References* to the *Object* instances of *StateTypes*, *TransitionTypes*, *EventTypes* and the *Methods* that collectively define the *Program FiniteStateMachine.*

Copyright International Electrotechnical Commission

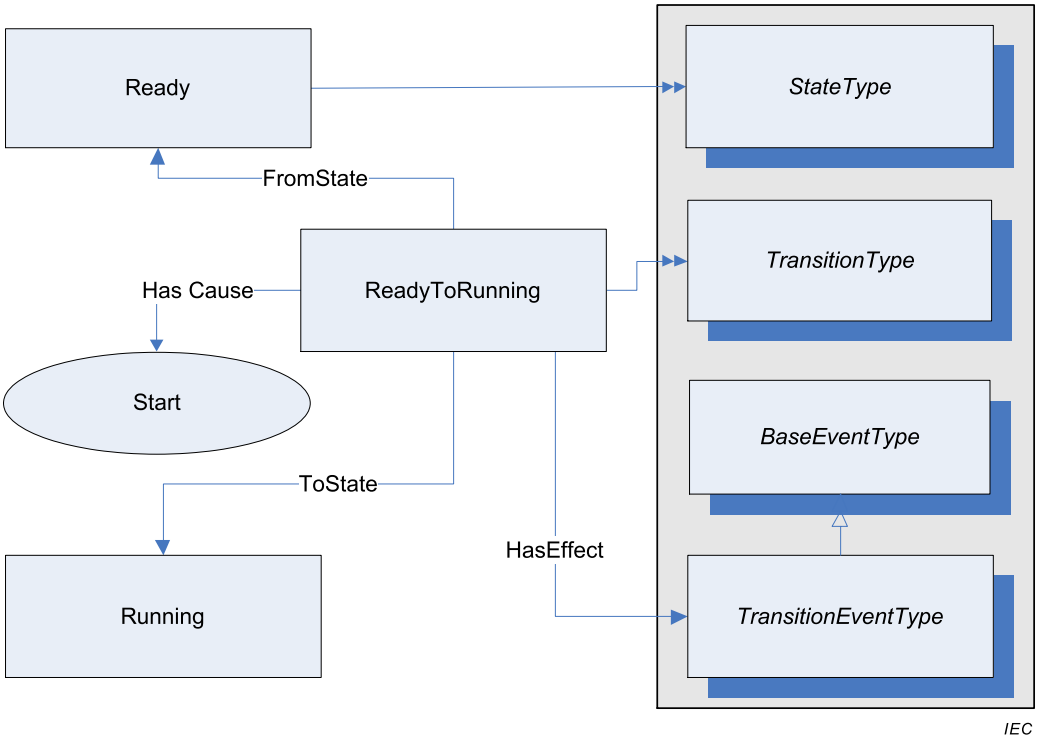

Figure 5 – Program FSM References

Figure 5 illustrates the component *References* that define the associations between two of the *ProgramType's* states, Ready and Running. The complementary ReferenceTypes have been omitted to simplify the illustration.

5.2.3.2 ProgramType states

Table 6 specifies the *ProgramType's* state *Objects*. These *Objects* are instances of the *StateType* defined in IEC 62541-5:2015, Annex B. Each state is assigned a unique *StateNumber* value. Subtypes of the *ProgramType* can add references from any state to a subordinate or nested *StateMachine Object* to extend the *FiniteStateMachine*.

Table 6 – Program states

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
States					
Halted	HasProperty	StateNumber	1	PropertyType	
	ToTransition	HaltedToReady		TransitionType	
	FromTransition	RunningToHalted		TransitionType	
	FromTransition	SuspendedToHalted		TransitionType	
	FromTransition	ReadyToHalted		TransitionType	
Ready	HasProperty	StateNumber	2	PropertyType	
	FromTransition	HaltedToReady		TransitionType	
	ToTransition	ReadyToRunning		TransitionType	
	FromTransition	RunningToReady		TransitionType	
	ToTransition	ReadyToHalted		TransitionType	

Copyright International Electrotechnical Commission

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Running	HasProperty	StateNumber	3	PropertyType	
	ToTransition	RunningToHalted		TransitionType	
	ToTransition	RunningToReady		TransitionType	
	ToTransition	RunningToSuspended		TransitionType	
	FromTransition	ReadyToRunning		TransitionType	
	FromTransition	SuspendedToRunning		TransitionType	
Suspended	HasProperty	StateNumber	4	PropertyType	
	ToTransition	SuspendedToRunning		TransitionType	
	ToTransition	SuspendedToHalted		TransitionType	
	ToTransition	SuspendedToReady		TransitionType	
	FromTransition	RunningToSuspended		TransitionType	

The Halted state is the idle state for a *Program*. It can be an initial state or a terminal state. As an initial state, the *Program Invocation* cannot begin execution due to conditions at the *Server*. As a terminal state, Halted can indicate either a failed or completed *Program*. A subordinate state or result can be used to distinguish the nature of the termination. The Halted state references four *Transition Objects*, which identify the allowed state transitions to the Ready state and from the Ready, Running, and Suspended states.

The Ready state indicates that the *Program* is prepared to begin execution. *Programs* that are ready to begin upon their creation may transition immediately to the Ready state. The Ready state references four *Transition Objects,* which identify the allowed state transitions to the Running and Halted states and from the Halted and Ready states.

The Running state indicates that the *Program* is actively performing its function. The Running state references five *Transition Objects,* which identify the allowed state transitions to the Halted, Ready, and Suspended states and from the Ready and Suspended states.

The Suspended state indicates that the *Program* has stopped performing its function, but retains the ability to resume the function at the point at which it was executing when suspended. The Suspended state references four *Transition Objects*, which identify the allowed state transitions to the Ready, Running, and Halted state and from the Ready state.

5.2.3.3 ProgramType transitions

ProgramType Transitions are instances of the *TransitionType* defined in IEC 62541-5:2015, Clause B.4 which also includes the definitions of the ToState, FromState, HasCause, and HasEffect references used. Table 7 specifies the transitions defined for the *ProgramType.* Each transition is assigned a unique *TransitionNumber*. The Notes column indicates when a cause is referencing *Methods* and when effects are optional.

Copyright International Electrotechnical Commission

Table 7 – Program transitions

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Transitions					
HaltedToReady	HasProperty	TransitionNumber	1	PropertyType	
	ToState	Ready		StateType	
	FromState	Halted		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToRunning	HasProperty	TransitionNumber	2	PropertyType	
	ToState	Running		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToHalted	HasProperty	TransitionNumber	3	PropertyType	
	ToState	Halted		StateType	
	FromState	Running		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToReady	HasProperty	TransitionNumber	4	PropertyType	
	ToState	Ready		StateType	
	FromState	Runnning		StateType	
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToSuspended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToRunning	HasProperty	TransitionNumber	6	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	

Copyright International Electrotechnical Commission

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Transitions					
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToHalted	HasProperty	TransitionNumber	7	PropertyType	
	ToState	Halted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToReady	HasProperty	TransitionNumber	8	PropertyType	
	ToState	Ready		StateType	
	FromState	Suspended		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToHalted	HasProperty	TransitionNumber	9	PropertyType	
	ToState	Halted		StateType	
	FromState	Ready		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			

The *HaltedToReady* transition specifies the transition from the Halted to Ready states. It may be caused by the *Reset Method.*

The *ReadyToRunning* transition specifies the transition from the Ready to *Running* states. It is caused by the *Start Method.*

The *RunningToHalted* transition specifies the transition from the *Running* to Halted states. It is caused by the *Halt Method.*

The *RunningToReady* transition specifies the transition from the Running to Ready states. The *RunningToSuspended* transition specifies the Transition from the Running to Suspended states. It is caused by the *Suspend Method.*

The *SuspendedToRunning* transition specifies the transition from the Suspended to Running states. It is caused by the *Resume Method.*

The *SuspendedToHalted* transition specifies the transition from the Suspended to Halted states. It is caused by the *Halt Method.*

Copyright International Electrotechnical Commission

The *SuspendedToReady* transition specifies the transition from the Suspended to Ready states. It is caused internally.

The *ReadyToHalted* transition specifies the transition from the Ready to Halted states. It is caused by the *Halt Method.*

Two *HasEffect* references are specified for each *Program* transition. These effects are *Events* of *ProgramTransitionEventType* and *AuditProgramTransitionEventType* defined in 5.2.5. The *ProgramTransitionEventType* notifies *Clients* of the *Program* transition and conveys result data. The *AuditProgramTransitionEventType* is used to audit transitions that result from *Program Control Methods*. The "*Audit Server Facet*" *Profile* defined in IEC 62541-7 requires support of the *AuditProgramTransitionEventType*.

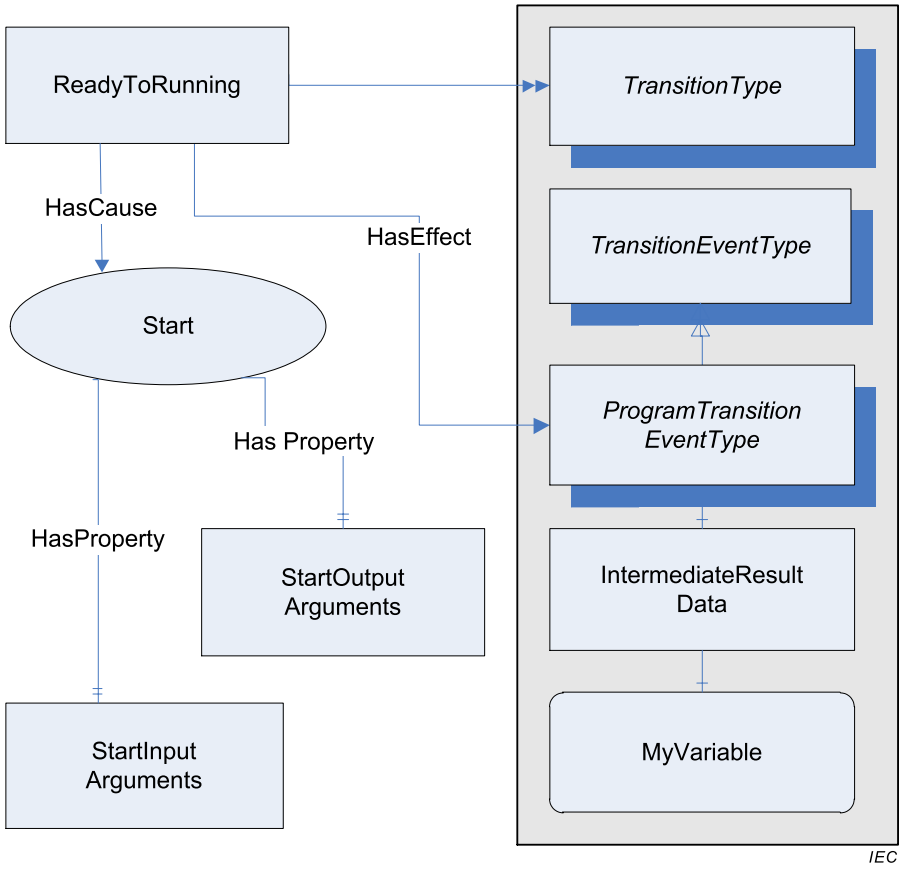

Figure 6 – ProgramType causes and effects

5.2.4 ProgramType causes (Methods)

5.2.4.1 Overview

The *ProgramType* includes references to the Causes of specific *Program* state transitions. These causes refer to *Method* instances. *Programs* that do not support *a Program Control Method* will omit the Causes reference to that *Method* from the *ProgramType* references. If a *Method's Causes* reference is omitted from the *ProgramType* then a *Client* cannot cause the associated state transition. The *Method* instances referenced by the *ProgramType* identify the *InputArguments* and *OutputArguments* required for the *Method* calls to *Program Invocations* of that *ProgramType*. Table 8 specifies the *Methods* defined as Causes for *ProgramTypes*. Figure 6 illustrates the *References* associating the components and *Properties* of *Methods* and *Events* with Program transitions.

Copyright International Electrotechnical Commission

Table 8 – ProgramType causes

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Causes					
Start	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Suspend	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Resume	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Halt	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Reset	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional

The *Start Method* causes the *ReadyToRunning Program* transition.

The *Suspend Method* causes the *RunningToSuspended Program* transition.

The *Resume Method* causes the *SuspendedToRunning Program* transition.

The *Halt Method* causes the *RunningToHalted, SuspendedToHalted, or ReadyToHalted Program* transition depending on the current state of the *Program*.

The *Reset Method* causes the *HaltedToReady Program* transition.

5.2.4.2 Standard attributes

The *Executable Method* attribute indicates if a method can currently be executed. For *Program Control Methods*, this means that the owning *Program* has a current state that supports the transition caused by the *Method.*

5.2.4.3 Standard properties

5.2.4.3.1 Overview

Methods can reference a set of *InputArguments.* For each *ProgramType*, a set of *InputArguments* may be defined for the supported *Program Control Methods*. The data passed in the arguments supplements the information required by the *Program* to perform its function. All calls to a *Program Control Method* for each *Program Invocation* of that *ProgramType* shall pass the specified arguments.

Methods can reference a set of *OutputArguments*. For each *ProgramType*, a set of *OutputArguments* is defined for the supported *Program Control Methods*. All calls to a *Program Control Method* for each *Program Invocation* of that *ProgramType* shall pass the specified arguments.

Copyright International Electrotechnical Commission

5.2.5 ProgramType effects (Events)

5.2.5.1 Overview

The *ProgramType* includes component references to the Effects of each of the *Program's* state transitions. These Effects are *Events.* Each *Transition* shall have a *HasEffect Reference* to a *ProgramTransitionEventType* and can have an *AuditProgramTransitionEventType.* When the transition occurs, *Event* notifications of the referenced type are generated for subscribed *Clients.* The *Program Invocation* may serve as the *EventNotifier* for these *Events* or an owning *Object* or the *Server Object* may provide the notifications.

ProgramTransitionEventTypes provide the means for delivering result data and confirming state transitions for subscribed *Clients* on each defined *Program State Transition.* The *AuditProgramTransitionEventType* allows the auditing of changes to the *Program*'s state in conjunction with *Client Method Calls.*

5.2.5.2 ProgramTransitionEventType

The *ProgramTransitionEventType* is a subtype of the *TransitionEventType*. It is used with *Programs* to acquire intermediate or final results or other data associated with a state transition. A *Program* can have a unique *ProgramTransitionEventType* definition for any transition. Each *ProgramTransitionEventType* specifies the *IntermediateResult* data specific to the designated state transition on that *ProgramType*. Each transition can yield different intermediate result data. Table 9 specifies the *ProgramTransitionEventType*.

Table 10 identifies the *ProgramTransitionEventTypes* that are specified for *ProgramTypes*.

Table 9 – ProgramTransitionEventType

Attribute	Value				
BrowseName	ProgramTransitionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the base *TransitionEventType* defined in IEC 62541-5:2015, B.4.16.					
HasComponent	Object	IntermediateResult		BaseObjectType	Optional

TransitionNumber identifies the *Program* transition that triggered the *Event*.

FromStateNumber identifies the state before the *Program* transition.

ToStateNumber identifies the state after the *Program* transition.

The *IntermediateResult* is an *Object* that aggregates a set of *Variables* whose values are relevant for the *Program* at the instant of the associated transition. The *ObjectType* for the *IntermediateResult* specifies the collection of *Variables* using a set of *HasComponent References*.

Copyright International Electrotechnical Commission

Table 10 – ProgramTransitionEvents

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Effects					
HaltedToReadyEvent					
	HasProperty	TransitionNumber	1	PropertyType	
	HasProperty	FromStateNumber	1	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
ReadyToRunningEvent					
	HasProperty	TransitionNumber	2	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToHaltedEvent					
	HasProperty	TransitionNumber	3	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToReadyEvent					
	HasProperty	TransitionNumber	4	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToSuspendedEvent					
	HasProperty	TransitionNumber	5	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	4	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToRunningEvent					
	HasProperty	TransitionNumber	6	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToHaltedEvent					
	HasProperty	TransitionNumber	7	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	

Copyright International Electrotechnical Commission

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Effects					
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToReadyEvent					
	HasProperty	TransitionNumber	8	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
ReadyToHaltedEvent					
	HasProperty	TransitionNumber	9	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional

5.2.6 AuditProgramTransitionEventType

The *AuditProgramTransitionEventType* is a subtype of the *AuditUpdateStateEventType*. It is used with *Programs* to provide a means to audit the *Program State* transitions associated with any *Client* invoked *Program Control Method*. *Servers* shall generate *AuditProgramTransitionEvents* if they support the *Audit Server Facet Profile* defined in IEC 62541-7.

Table 11 specifies the definition of the *AuditProgramTransitionEventType.*

Table 11 – AuditProgramTransitionEventType

Attribute	Value				
BrowseName	AuditProgramTransitionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateStateEventType* defined in IEC 62541-5:–, B.4.17.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateStateEventType* defined in IEC 62541-5:2015, B.4.17, except as noted below.

The *Status Property,* specified in IEC 62541-5:2015, 6.4.3,, identifies whether the state transition resulted from a Program Control Method call (set *Status* to TRUE) or not (set *Status* to FALSE).

The *SourceName* specified in IEC 62541-5:2015, 6.4.2, identifies the *Method* causing the *Program* transition when it is the result of a *Client* invoked *ProgramControlMethod*. The *SourceName* is prefixed with “Method/” and the name of the *ProgramControlMethod*, “Method/Start” for example.

The *ClientUserId Property,* specified in IEC 62541-5:2015, 6.4.3, identifies the user of the *Client* that issued the *Program Control Method* if it is associated with this *Program* state transition.

Copyright International Electrotechnical Commission

The *ActionTimeStamp Property,* specified in IEC 62541-5:2015, 6.4.3 “AuditEventType”, identifies when the time the *Program* state transition that resulted in the *Event* being generated occurred.

The *TransitionNumber Property* is a *Variable* that identifies the transition that triggered the *Event*.

5.2.7 FinalResultData

The *FinalResultData ObjectType* specifies the *VariableTypes* that are preserved when the *Program* has completed its function. The *ObjectType* includes *a HasComponent* for a *VariableType* of each *Variable* that comprises the final result data.

5.2.8 ProgramDiagnostic DataType

This structure contains elements that chronicle the *Program Invocation’s* activity and can be used to aid in the diagnosis of *Program* problems. Its composition is defined in Table 12.

Table 12 – ProgramDiagnosticDataType structure

Name	Type	Description
ProgramDiagnosticDataType	structure	
createSessionId	NodeId	The *CreateSessionId* contains the *SessionId* of the *Session*on which the call to the *Create Method* was issued to create the *Program Invocation*.
createClientName	String	The *CreateClientName* is the name of the client of the *Session*that created the *Program Invocation*.
invocationCreationTime	UTCTime	The *InvocationCreationTime* identifies the time the *Program Invocation* was created.
lastTransitionTime	UTCTime	The *LastTransitionTime* identifies the time of the last *Program* state transition that occurred.
lastMethodCall	String	The *LastMethodCall* identifies the last *Program Method* called on the *Program Invocation*.
lastMethodSessionId	NodeId	The *LastMethodSessionId* contains the *SessionId* of the *Session*on which the last Program Control Method call to the *Program Invocation* was issued.
lastMethodInputArguments	Argument	The *LastMethodInputArguments* preserves the values of the input arguments on the last *Program Method* call.
lastMethodOutputArguments	Argument	The *LastMethodOutputArguments* preserves the values of the output arguments on the last *Program Method* call.
lastMethodCallTime	UTCTime	The *LastMethodCallTime* identifies the time of the last Method call to the *Program Invocation*.
lastMethodReturnStatus	StatusResult	The *LastMethodReturnStatus* preserves the value of the return status for the last *Program Control Method* requested for *this Program Invocation.*

Its representation in the *AddressSpace* is defined in Table 13.

Table 13 – ProgramDiagnosticDataType definition

Attributes	Value
BrowseName	ProgramDiagnosticDataType

Copyright International Electrotechnical Commission

5.2.9 ProgramDiagnosticType VariableType

This *VariableType* aggregates simple *Variables* using simple *DataTypes* that reflect the elements of the ProgramDiagnosticDataType structure. Its *DataVariables* have the same semantic as defined in in 5.2.8. The *VariableType* is formally defined in Table 14.

Table 14 – ProgramDiagnosticType VariableType

Attribute	Value			
BrowseName	ProgramDiagnosticType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in IEC 62541-5.				
HasComponent	Variable	CreateSessionId	NodeId	Mandatory
HasComponent	Variable	CreateClientName	String	Mandatory
HasComponent	Variable	InvocationCreationTime	UTCTime	Mandatory
HasComponent	Variable	LastTransitionTime	UTCTime	Mandatory
HasComponent	Variable	LastMethodCall	String	Mandatory
HasComponent	Variable	LastMethodSessionId	NodeId	Mandatory
HasComponent	Variable	LastMethodInputArguments	Argument	Mandatory
HasComponent	Variable	LastMethodOutputArguments	Argument	Mandatory
HasComponent	Variable	LastMethodCallTime	UTCTime	Mandatory
HasComponent	Variable	LastMethodReturnStatus	StatusResult	Mandatory

Copyright International Electrotechnical Commission

Annex A
(informative)

Program example

A.1 Overview

This example illustrates the use of a *Program* to manage a domain download into a control system as depicted in Figure A.1. The download requires the segmented transfer of control operation data from a secondary storage device to the local memory within a control system.

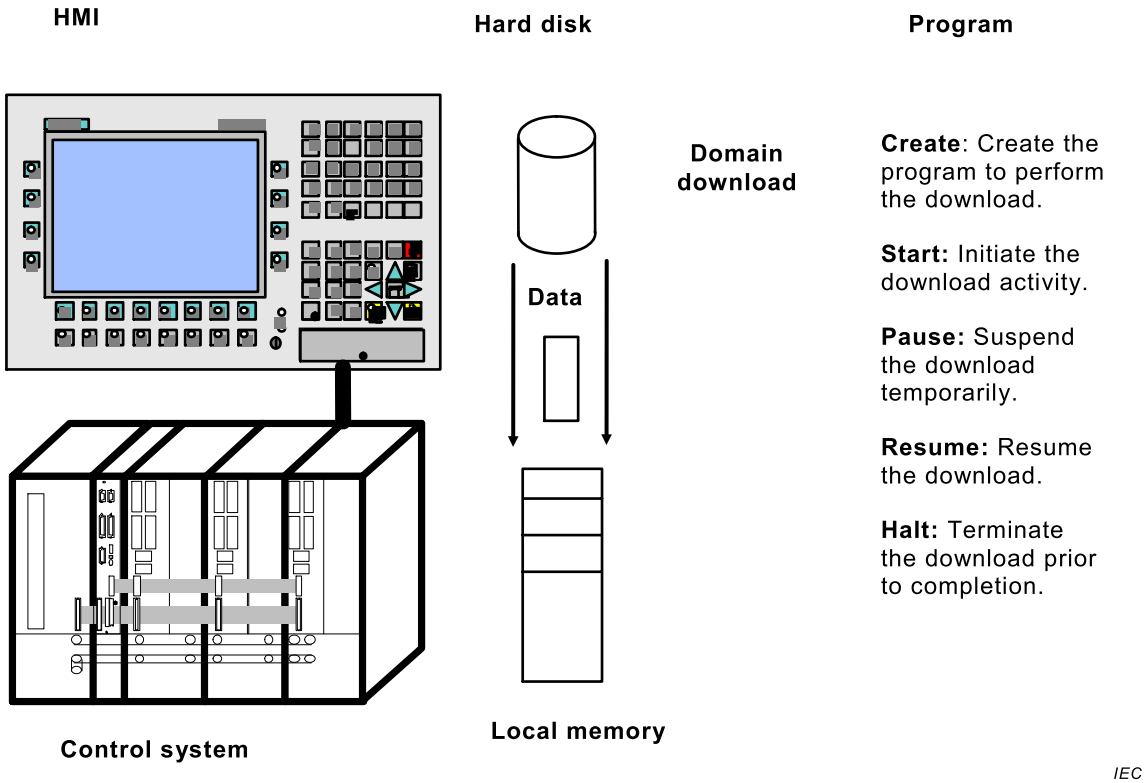

Figure A.1 – Program example

The domain download has a source and a target location which are identified when the download is initiated. Each time a segment of the domain is successfully transferred the client is notified and informed of the amount of data that has been downloaded. The client is also notified when the download is finished. The percentage of the total data received is reported periodically while the download continues. If the download fails, the cause of the failure is reported. At the completion of the download, the performance information is kept at the *Server*.

A.2 DomainDownload Program

A.2.1 General

The *Client* uses the “DomainDownload” *Program* to manage and monitor the download of a domain at the *Server*.

Copyright International Electrotechnical Commission

A.2.2 DomainDownload states

The basic state model for the DomainDownload *Program* is presented in Figure A.2. The *Program* has three primary states, Ready, Running, and Halted which are aligned with the standard states of a *ProgramType*. Additionally, the *DomainDownloadType* extends the *ProgramType* by defining subordinate state machines for the *Program's* Running and Halted states. The subordinate states describe the download operations in greater detail and allow the *Client* to monitor the activity of the download at a finer resolution.

An instance (*Program Invocation*) of a DomainDownload *Program* is created by the *Client* each time a download is to be performed. The instance exists until explicitly removed by the *Client*. The initial state of the *Program* is Ready and the terminal state is Halted. The DomainDownload can be temporarily suspended and then resumed or aborted. Once halted, the program may not be restarted.

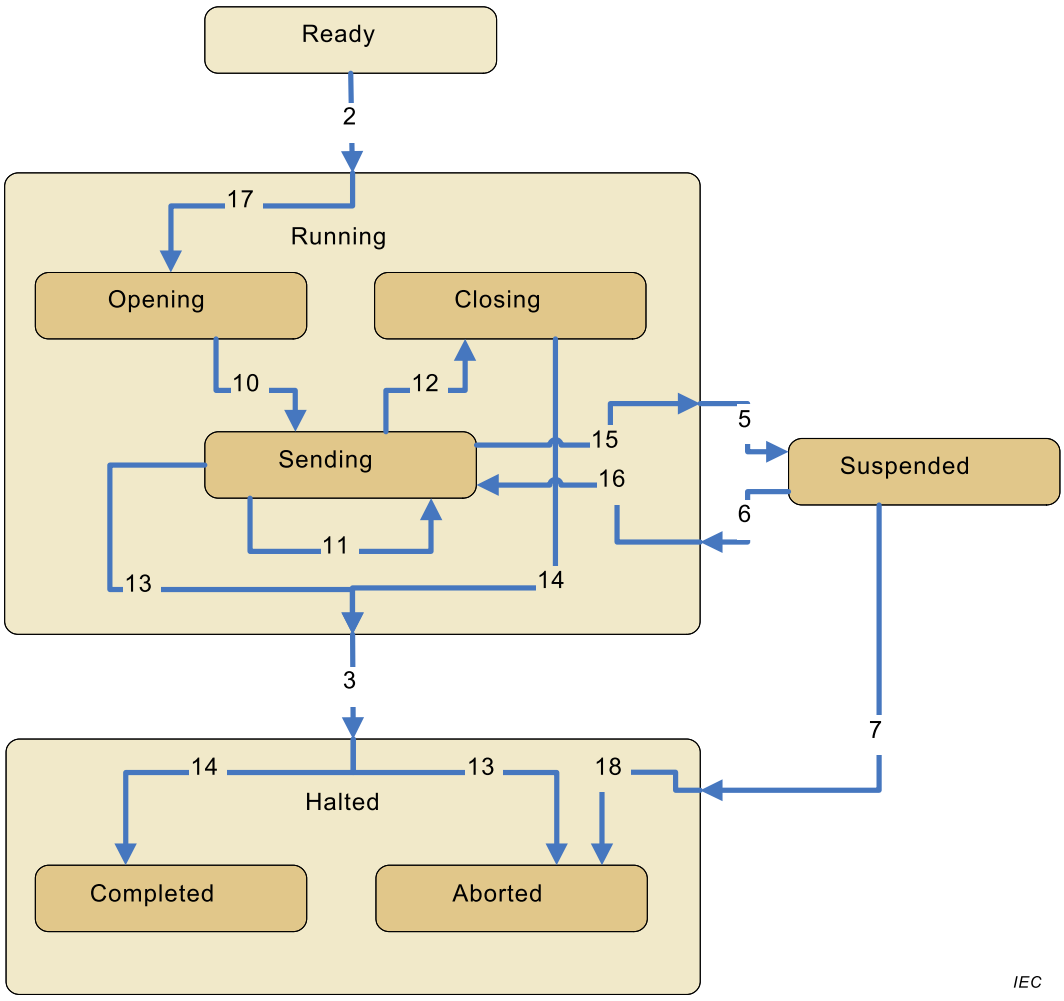

Figure A.2 – DomainDownload state diagram

The sequence of state transitions is illustrated in Figure A.2. Once the download is started, The *Program* progresses to the Opening state. After the source of the data is opened, a sequence of transfers occurs in the Sending state. When the transfer completes the *Objects* are closed in the Closing state. If the transfer is terminated before all of the data is downloaded or an error is encountered then the download is halted and the *Program* transitions to the Aborted state; otherwise the *Program* halts in the Completed state. The states are presented in Table A.1 along with the state transitions.

Copyright International Electrotechnical Commission

A.2.3 DomainDownload transitions

The valid state transitions specified for the DomainDownload *Program* are specified in Table A.1. Each of the transitions defines a start state and end state for the transition and is identified by a unique number. Five of the transitions are from the base *ProgramType* and retain the transition identifier numbers specified for *Programs*. The additional transitions relate the base *Program* states with the subordinate states defined for the DomainDownload. These states have been assigned unique transition identifier numbers that distinguish them from the base *Program* transition identifiers. In cases where transitions occur between substates and the *Program's* base states, two transitions are specified. One transition identifies the base state change and a second sub-state change. For example, Ready to Running and to Opening occurs at the same time.

The table also specifies the defined states, causes for the transitions, and the effects of each transition. *Program Control Methods* are used by the *Client* to "run" the DomainDownload. The *Methods* cause or trigger the specified transitions. The transition effects are the specified *EventTypes* which notify the *Client* of *Program* activity.

Table A.1 – DomainDownload states

No.	Transition name	Cause	From State	To State	Effect
2	ReadyToRunning	Start Method	Ready	Running	Report Transition 2 Event/Result
3	RunningToHalted	Halt Method/Error or Internal.	Running	Halted	Report Transition 3 Event/Result
5	RunningToSuspended	Suspend Method	Running	Suspended	Report Transition 5 Event/Result
6	SuspendedToRunning	Resume Method	Suspended	Running	Report Transition 6 Event/Result
7	SuspendedToHalted	Halt Method	Suspended	Halted	Report Transition 7 Event/Result
10	OpeningToSending	Internal	Opening	Sending	Report Transition 10 Event/Result
11	SendingToSending	Internal	Sending	Sending	Report Transition 11 Event/Result
12	SendingToClosing	Internal	Sending	Closing	Report Transition 12 Event/Result
13	SendingToAborted	Halt Method/Error	Opening	Aborted	Report Transition 13 Event/Result
14	ClosingToCompleted	Internal	Closing	Completed	Report Transition 14 Event/Result
15	SendingToSuspended	Suspend Method	Sending	Suspended	Report Transition 16 Event/Result
16	SuspendedToSending	Resume Method	Suspended	Sending	Report Transition 17 Event/Result
18	SuspendedToAborted	Halt Method	Suspended	Aborted	Report Transition 18 Event/Result
17	ToOpening	Internal	Ready	Opening	Report Transition 19 Event/Result

A.2.4 DomainDownload Methods

A.2.4.1 General

Four standard *Program Methods* are specified for running the DomainDownload *Program, Start, Suspend, Resume, and Halt.* No additional *Methods* are specified. The base behaviours of these *Methods* are defined by the *ProgramType.* The *Start Method* initiates the download activity and passes the source and destination locations for the transfer. The *Suspend Method* is used to pause the activity temporarily. The *Resume Method* reinitiates the download, when paused. The *Halt Method* aborts the download. Each of the *Methods* causes a *Program* state transition and a sub state transition. The specific state transition depends on the current state at the time the *Method* is called. If a *Method Call* is made when the DomainDownload is in a state for which that *Method* has no associated transition, the *Method* returns an error status indicating invalid state for the *Method*.

Copyright International Electrotechnical Commission

A.2.4.2 Method Arguments

The *Start Method* specifies three input arguments to be passed when it is called: Domain Name, DomainSource, and DomainDestination. The other *Methods* require no input arguments. No output arguments are specified for the DomainDownload *Methods*. The resultant error status for the *Program* is part of the *Call Service.*

A.2.5 DomainDownload Events

A.2.5.1 General

A *ProgramTransitionEventType* is specified for each of the DomainDownload *Program* transitions. The *EventTypes* trigger a specific *Event* notification to the *Client* when the associated state transition occurs in the running *Program* Instance. The *Event* notification identifies the transition. The SendingToSending state transition also includes intermediate result data.

A.2.5.2 Event information

The SendingToSending *Program* transition *Event* relays intermediate result data to the *Client* along with the notification. Each time the transition occurs, data items describing the amount and percentage of data transferred are sent to the *Client*.

A.2.5.3 Final result data

The DomainDownload *Program* retains final result data following a completed or aborted download. The data includes the total transaction time and the size of the domain. In the event of an aborted download, the reason for the termination is retained.

A.2.6 DomainDownload model

A.2.6.1 Overview

The OPC UA model for the DomainDownload *Program* is presented in the following tables and figures. Collectively they define the components that constitute this *Program*. For clarity, the figures present a progression of portions of the model that complement the contents of the tables and illustrate the *Program's* composition.

The type definition for the DomainDownload *Program* precisely represents the behaviour of the *Program* in terms of OPC UA components. These components can be browsed by a *Client* to interpret or validate the actions of the *Program*.

A.2.6.2 DomainDownloadType

The DomainDownloadType is a subtype derived from the *ProgramType*. It specifies the use or non-use of optional *ProgramType* components, valid extensions such as subordinate state machines, and constrained attribute values applied to instances of DomainDownload *Programs*.

Table A.2 specifies the optional and extended components defined by the DomainDownload Type. Note the references to two sub *State Machine Types*, *TransferStateMachine* and *FinishStateMachine*. The DomainDownloadType omits references to the *Reset Program Control Method* and its associated state transition (HaltedToReady), which it does not support.

Copyright International Electrotechnical Commission

Table A.2 – DomainDownload Type

Attribute	Value				
	Includes all non-optional attributes specified for the ProgramType				
BrowseName	DomainDownloadType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	TransferStateMachine		StateMachineType	Mandatory
HasComponent	Object	FinishStateMachine		StateMachineType	Mandatory
HasComponent	Variable	ProgramDiagnostic		ProgramDiagnosticType	Mandatory
HasComponent	Object	ReadyToRunning		TransitionType	Mandatory
HasComponent	Object	RunningToHalted		TransitionType	Mandatory
HasComponent	Object	RunningToSuspended		TransitionType	Mandatory
HasComponent	Object	SuspendedToRunning		TransitionType	Mandatory
HasComponent	Object	SuspendedToHalted		TransitionType	Mandatory
HasComponent	Method	Start			Mandatory
HasComponent	Method	Suspend			Mandatory
HasComponent	Method	Halt			Mandatory
HasComponent	Method	Resume			Mandatory
HasComponent	Object	FinalResultData		BaseObjectType	Mandatory

Table A.3 specifies the *Transfer State Machine type* that is a sub state machine of the DomainDownload *Program Type*. This definition identifies the *StateTypes* that compose the sub states for the *Program's* Running *StateType*.

Table A.3 – Transfer State Machine Type

Attribute	Value				
	Includes all attributes specified for the FiniteStateMachineType				
BrowseName	TransferStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Opening		StateType	Mandatory
HasComponent	Object	Sending		StateType	Mandatory
HasComponent	Object	Closing		StateType	Mandatory
HasComponent	Object	ReadyToOpening		TransitionType	Mandatory
HasComponent	Object	OpeningToSending		TransitionType	Mandatory
HasComponent	Object	SendingToClosing		TransitionType	Mandatory
HasComponent	Object	SendingToAborted		TransitionType	Mandatory
HasComponent	Object	SendingToSuspended		TransitionType	Mandatory
HasComponent	Object	SuspendedToSending		TransitionType	Mandatory
HasComponent	Method	Start			Mandatory
HasComponent	Method	Suspend			Mandatory
HasComponent	Method	Halt			Mandatory
HasComponent	Method	Resume			Mandatory

Table A.3 specifies the *StateTypes* associated with the Transfer State Machine Type. All of these states are sub states of the *Running* state of the base *ProgramType*.

The Opening state is the preparation state for the domain download.

Copyright International Electrotechnical Commission

The Sending state is the activity state for the transfer in which the data is moved from the source to destination.

The Closing state is the cleanup phase of the download.

Table A.4 defines the states of the *TransferStateMachineType*.

Table A.4 – Transfer State Machine – states

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
States					
Opening	HasProperty	StateNumber	5	PropertyType	
	ToTransition	OpeningToSending		TransitionType	
	FromTransition	ToOpening		TransitionType	
	ToTransition	OpeningToSending		TransitionType	
Sending	HasProperty	StateNumber	6	PropertyType	
	FromTransition	OpeningToSending		TransitionType	
	ToTransition	SendingToSending		TransitionType	
	ToTransition	SendingToClosing		TransitionType	
	ToTransition	SendingToSuspended		TransitionType	
	FromTransition	ToSending		TransitionType	
Closing	HasProperty	StateNumber	7	PropertyType	
	ToTransition	ClosingToCompleted		TransitionType	
	ToTransition	ClosingToAborted		TransitionType	
	FromTransition	SendingToClosing		TransitionType	

Table A.5 specifies the *Finish State Machine Type* that is a sub state machine of the DomainDownload *ProgramType*. This definition identifies the *StateTypes* that compose the sub states for the *Program's* Halted *StateType*.

Table A.5 – Finish State Machine Type

Attribute	Value				
	Includes all attributes specified for the FiniteStateMachineType				
BrowseName	TransferStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Completed		StateType	Mandatory
HasComponent	Object	Aborted		StateType	Mandatory

Table A.6 specifies the *StateTypes* associated with the *Finish State Machine Type*. Note these are final states and that they have no associated transitions between them.

Copyright International Electrotechnical Commission

Table A.6 – Finish State Machine – states

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
States					
Aborted	HasProperty	StateNumber	8	PropertyType	
	FromTransition	OpeningToAborted		TransitionType	
	FromTransition	ClosingToAborted		TransitionType	
Completed	HasProperty	StateNumber	9	PropertyType	
	FromTransition	ClosingToCompleted		TransitionType	

The Aborted state is the terminal state that indicates an incomplete or failed domain download operation.

The Completed state is the terminal state that indicates a successful domain download.

Table A.7 specifies the constraining behaviour of a DomainDownload.

Table A.7 – DomainDownload Type Property Attributes variable values

NodeClass	BrowseName	Data Type	Data Value	Modelling Rule
Variable	Creatable	Boolean	True	--
Variable	Deletable	Boolean	True	Mandatory
Variable	AutoDelete	Boolean	False	Mandatory
Variable	RecycleCount	Int32	0	Mandatory
Variable	InstanceCount	UInt32	PropertyType	--
Variable	MaxInstanceCount	UInt32	500	--
Variable	MaxRecycleCount	UInt32	0	--

A DomainDownload *Program Invocation* can be created and also destroyed by a *Client*. The *Program Invocation* will not delete itself when halted, but will persist until explicitly removed by the *Client*. A DomainDownload *Program Invocation* cannot be reset to restart. The *Server* will support up to 500 concurrent DomainDownload *Program Invocation*s.

Figure A.3 presents a partial DomainDownloadType model that illustrates the association between the states and the DomainDownload, Transfer, and Finish State Machines. Note that the current state number for the sub state machines is only valid when the DomainDownload active base state references the sub state machine, Running for the Transfer current state and Halted for the Finish current state.

Copyright International Electrotechnical Commission

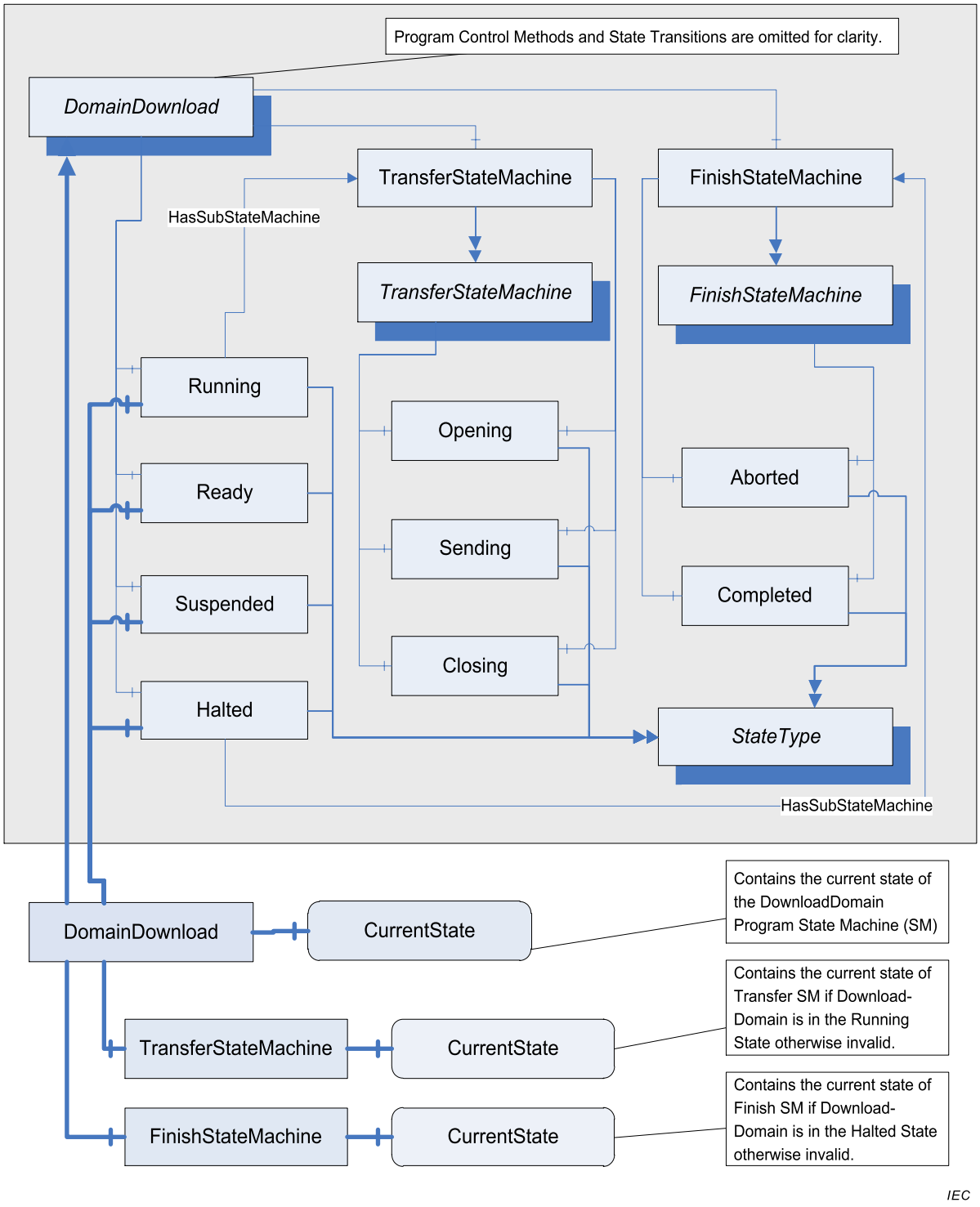

Figure A.3 – DomainDownloadType partial state model

Table A.8 specifies the *ProgramTransitionTypes* that are defined in addition to the *ProgramTransitionTypes* specified for *Programs* in Table 7. These types associate the Transfer and Finish sub state machine states with the states of the base *Program.*

Copyright International Electrotechnical Commission

Table A.8 – Additional DomainDownload transition types

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Transitions					
ToSending	HasProperty	TransitionNumber	10	PropertyType	
	ToState	Sending		StateType	
	FromState	Opening		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SendingToSending	HasProperty	TransitionNumber	11	PropertyType	
	ToState	Sending		StateType	
	FromState	Sending		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToClosing	HasProperty	TransitionNumber	12	PropertyType	
	ToState	Closing		StateType	
	FromState	Sending		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToAborted	HasProperty	TransitionNumber	13	PropertyType	
	ToState	Aborted		StateType	
	FromState	Sending		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ClosingToCompleted	HasProperty	TransitionNumber	14	PropertyType	
	ToState	Completed		StateType	
	FromState	Closing		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToSuspended	HasProperty	TransitionNumber	15	PropertyType	
	ToState	Suspended		StateType	
	FromState	Sending		StateType	
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToSending	HasProperty	TransitionNumber	16	PropertyType	
	ToState	Sending		StateType	
	FromState	Suspended		StateType	

Copyright International Electrotechnical Commission

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Transitions					
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToAborted	HasProperty	TransitionNumber	18	PropertyType	
	ToState	Aborted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToOpening	HasProperty	TransitionNumber	17	PropertyType	
	ToState	Opening		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			

Figure A.4 through Figure A.10 illustrate portions of the DomainDownloadType model. In each figure, the referenced tates, *Methods*, transitions, and *EventTypes* are identified for one or two state transitions.

Copyright International Electrotechnical Commission

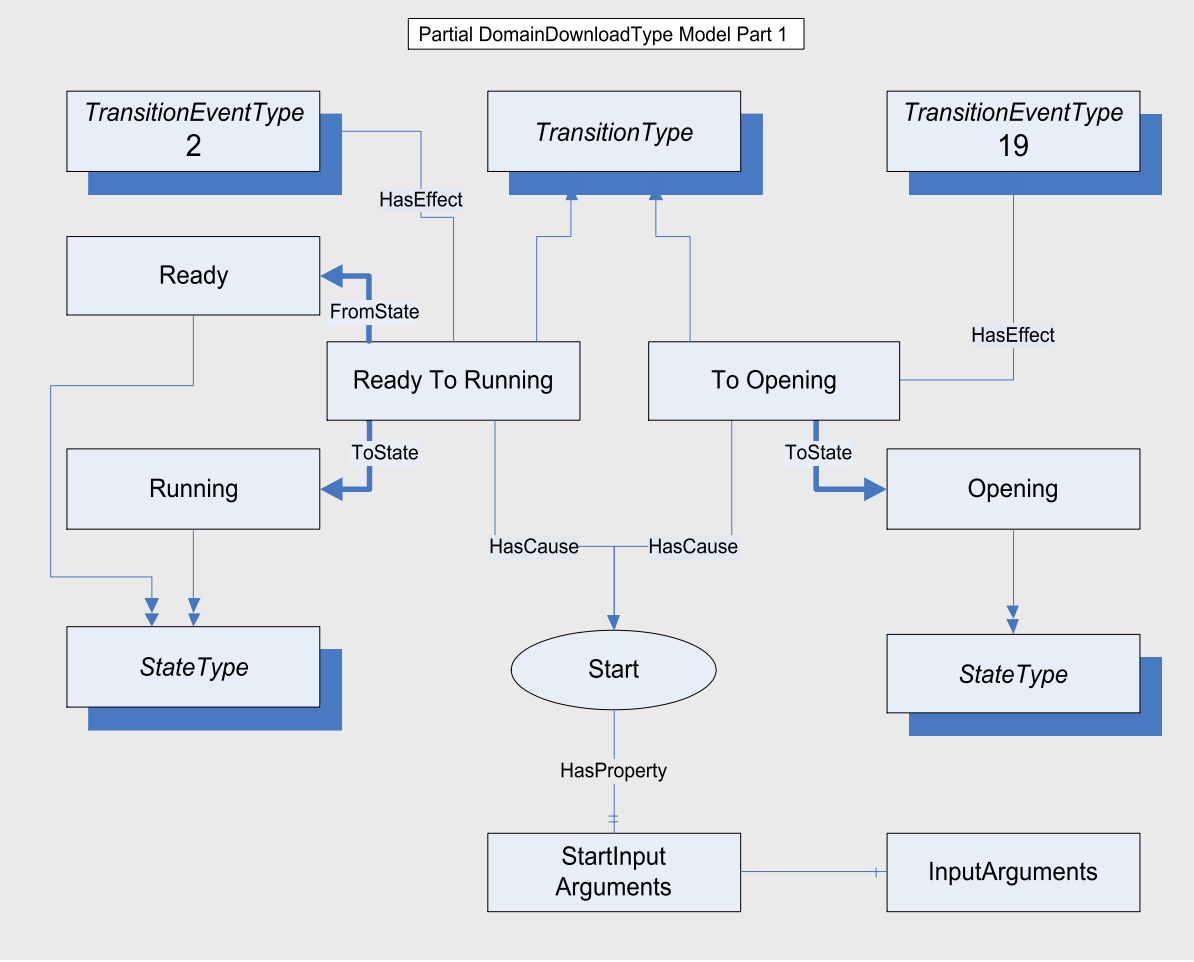

Figure A.4 – Ready To Running model

Figure A.4 illustrates the model for the ReadyToRunning *Program* transition. The transition is caused by the *Start Method*. The *Start Method* requires three input arguments. The *Method Call* service is used by the *Client* to invoke the *Start Method* and pass the arguments. When successful, the *Program Invocation* enters the Running state and the subordinate Transfer Opening state. The *Server* issues two *Event* notifications, ReadyToRunning (2), and ToOpening (19).

Table A.9 – Start Method additions

Attribute	**Value**				
BrowseName	Start				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArgument	Argument[]	PropertyType	–

Table A.9 specifies that the *Start Method* for the *DomainDownloadType* requires input arguments. Table A.10 identifies the *Start Arguments* required.

Copyright International Electrotechnical Commission

Table A.10 – StartArguments

Name	Type	Value
Argument 1	structure	
name	String	SourcePath
dataType	NodeId	StringNodeId
valueRank	Int32	-1 (-1 = scalar)
arrayDimensions	UInt32[]	null
description	LocalizedText	The source specifier for the domain
Argument 2	structure	
Name	String	DesinationPath
dataType	NodeId	StringNodeId
valueRank	Int32	-1 (-1 = scalar)
arrayDimensions	UInt32[]	null
description	LocalizedText	The destination specifier for the domain
Argument 3	structure	
name	String	DomainName
dataType	NodeId	StringNodeId
arrayDimensions	UInt32[]	null
valueRank	Int32	-1 (-1 = scalar)
description	LocalizedText	The name of the domain

Figure A.5 illustrates the model for the Opening To Sending and the Sending to Closing *Program* transitions. As specified in the transition table, these state transitions require no *Methods* to occur, but rather are driven by the internal actions of the *Server*. *Events* are generated for each state transition (10 to 12), when they occur.

Copyright International Electrotechnical Commission

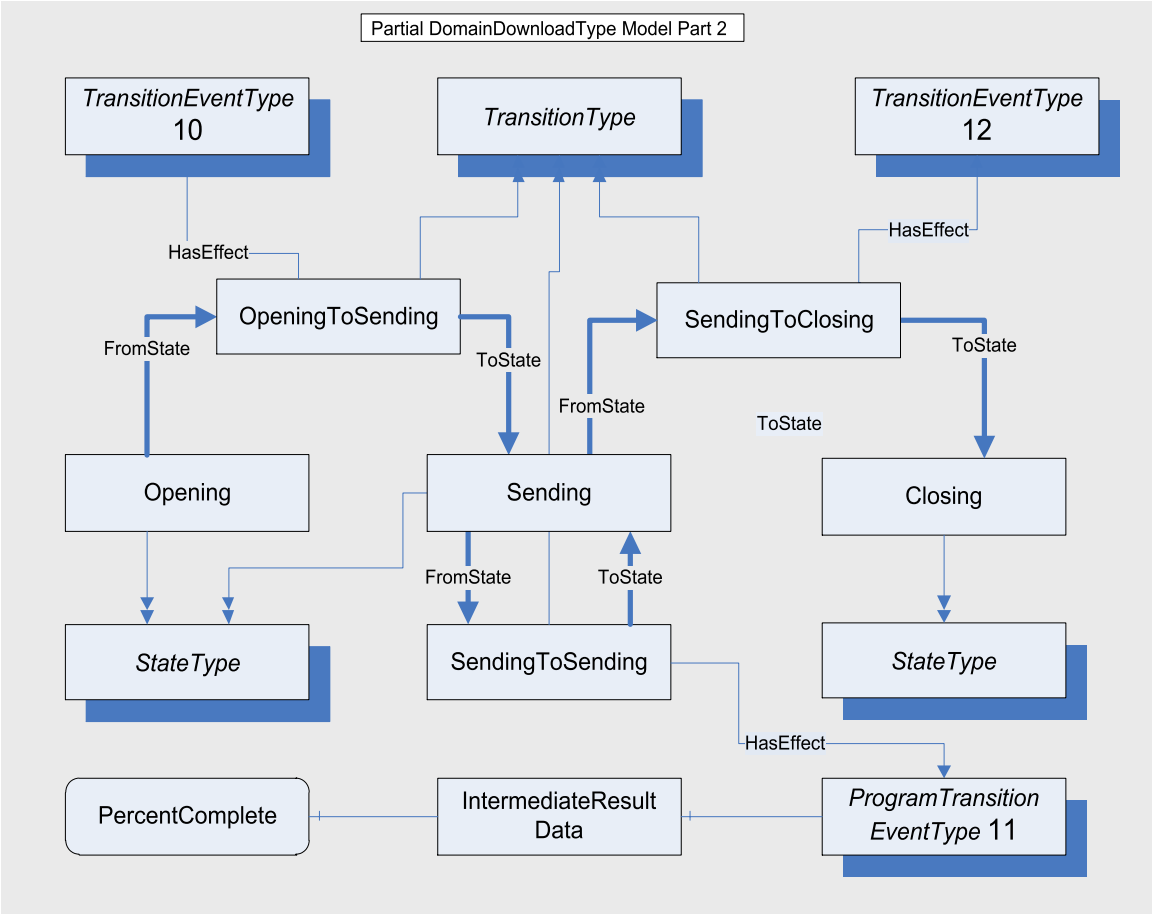

Figure A.5 – Opening To Sending To Closing model

Notice that a state transition can initiate and terminate at the same state (Sending). In this case the transition serves a purpose. The *ProgramTransitionEventType* effect referenced by the SendingToSending state transition has an *IntermediateResultData Object Reference*. The *IntermediateResultData Object* serves to identify two *Variables* whose values are obtained each time the state transition occurs. The values are sent to the *Client* with the *Event* notification. Table A.11 defines the *IntermediateResults ObjectType* and Table A.12 defines the *Variables* of the *ObjectType*.

Table A.11 – IntermediateResults Object

Attribute	**Value**				
	Includes all attributes specified for the ObjectType				
BrowseName	IntermediateResults				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	AmountTransferred	Long	VariableType	Mandatory
HasComponent	Variable	PercentageTransferred	Long	VariableType	Mandatory

Copyright International Electrotechnical Commission

Table A.12 – Intermediate result data Variables

Intermediate Result Variables	Type	Value
Variable 1	Structure	
Name	String	AmountTransferred
dataType	NodeId	StringNodeId
description	LocalizedText	Bytes of domain data transferred.
Variable 2	Structure	
Name	String	PercentageTransferred
dataType	NodeId	StringNodeId
description	LocalizedText	Percentage of domain data transferred.

The model for the Running To Suspended state transition is illustrated in Figure A.6. The cause for this transition is the *Suspend Method.* The *Client* can pause the download of domain data to the control. The transition from Running to Suspended invokes the *Event* generation for *TransitionEventTypes* 5 and 16. Note that there is no longer a valid current state for the Transfer State Machine.

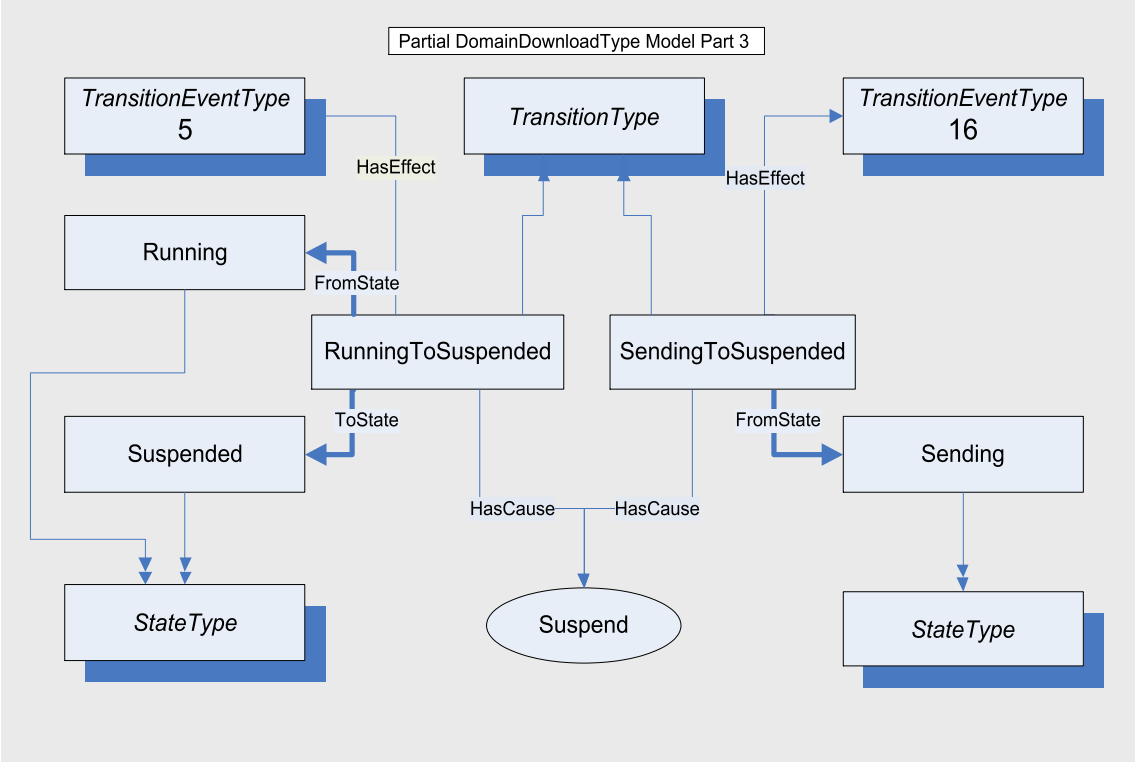

Figure A.6 – Running To Suspended model

The model for the SuspendedToRunning state transition is illustrated in Figure A.7. The cause for this transition is the *Resume Method.* The *Client* can resume the download of domain data to the control. The transition from Suspended to Running generates the *Event* for *TransitionEventTypes* 6 and 17. Now that the Running state is active, the Sending state of the Transfer State Machine is again specified for the *CurrentStateNumber*.

Copyright International Electrotechnical Commission

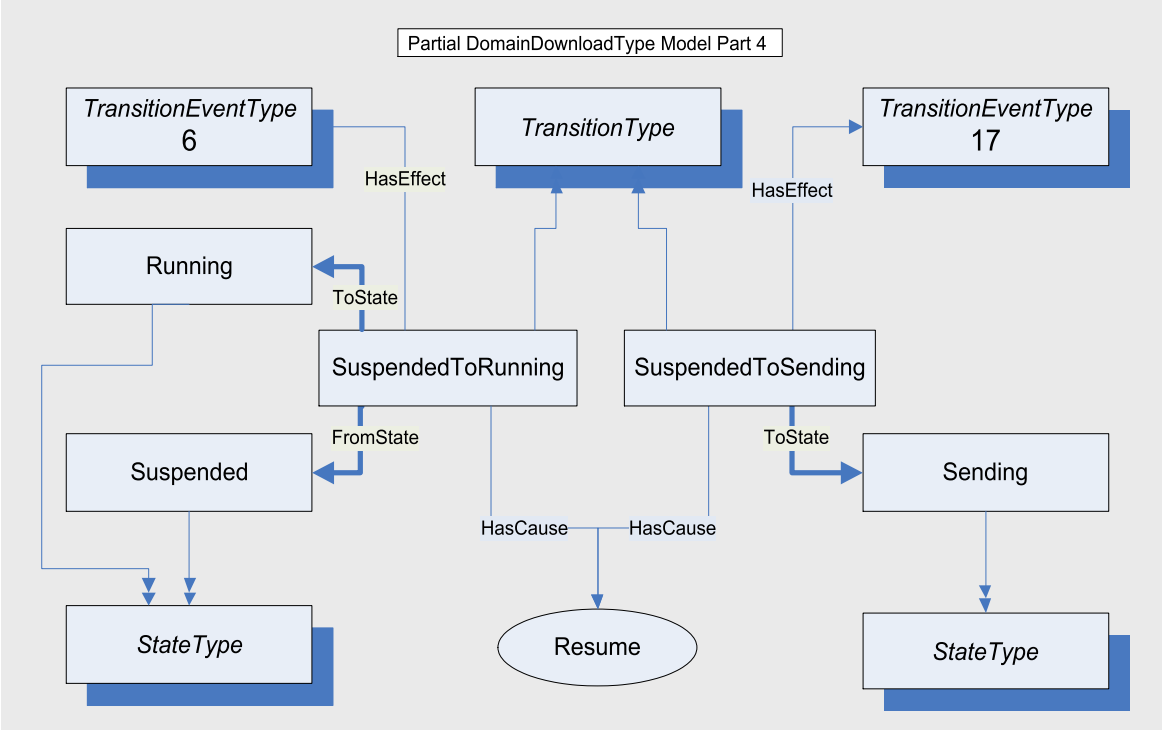

Figure A.7 – Suspended To Running model

The model for the Running To Halted state transition for an abnormal termination of the domain download is illustrated in Figure A.8. The cause for this transition is the *Halt Method.* The *Client* can terminate the download of domain data to the control. The transition from Running To Halted generates the *Event* for *TransitionEventTypes* 3 and 15. The *TransitionEventType* 15 indicates the transition from the Sending state as the Running State ends and then to the Aborted state as the Halted state is entered.

Copyright International Electrotechnical Commission

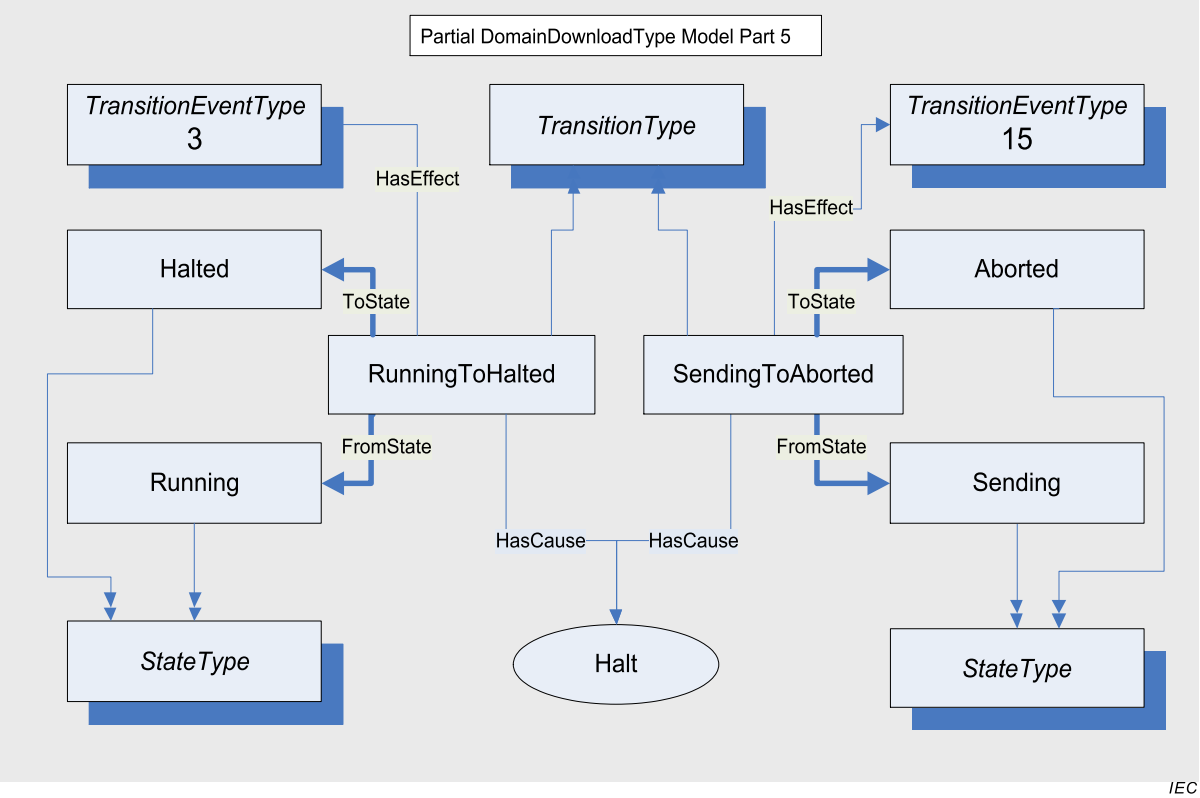

Figure A.8 – Running To Halted – Aborted model

Figure A.9 illustrates the model for the Suspended To Halted state transition for an abnormal termination of the domain download. The cause for this transition is the *Halt Method.* The *Client* can terminate the download of domain data to the control while it is suspended. The transition from SuspendedToHalted invokes the *Event* notifiers for *TransitionEventTypes* 7 and 18.

Copyright International Electrotechnical Commission

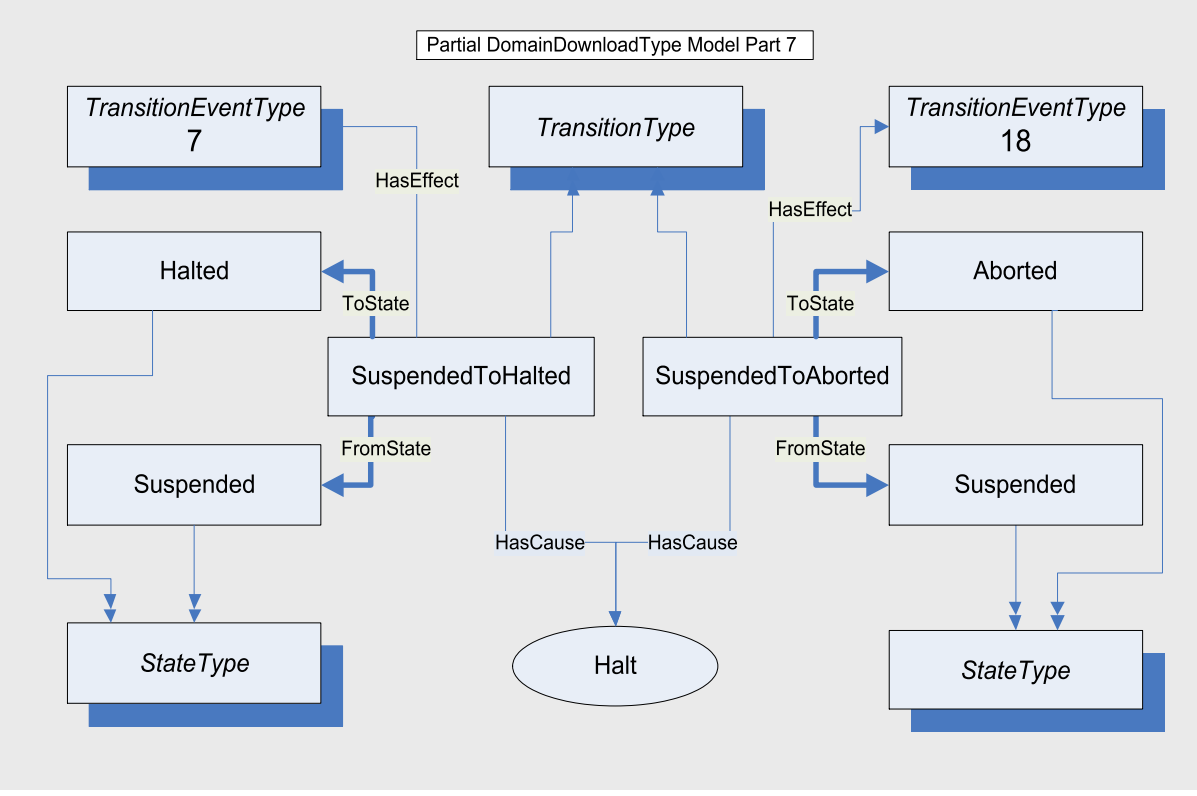

Figure A.9 – Suspended To Aborted model

The model for the Running To Completed state transition for a normal termination of the domain download is illustrated in Figure A.10. The cause for this transition is internal. The transition from Closing To Halted generates the Event for *TransitionEventTypes* 3 and 14. The *TransitionEventType* 14 indicates the transition from the Closing state as the Running state ends and then to the Completed state as the Halted state is entered.

The DomainDownloadType includes a component reference to a *FInalResultData Object*. This *Object* references *Variables* that persists information about the domain download once it has completed. This data can be read by *Clients* who are not subscribed to *Event* notifications. The result data is described in Table A.13 and the *Variables* in Table A.14.

Table A.13 – FinalResultData

Attribute	**Value**				
	Includes all attributes specified for the ObjectType				
BrowseName	FinalResultData				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	DownloadPerformance	Long	VariableType	Mandatory
HasComponent	Variable	FailureDetails	Long	VariableType	Mandatory

The Domain Download net transfer data rate and detailed reason for aborted downloads is retained as final result data for each *Program Invocation*.

Copyright International Electrotechnical Commission

Table A.14 – Final result Variables

Final Result Variables	Type	Value
Variable 1	Structure	
Name	String	DownloadPerformance
dataType	NodeId	Double
description	LocalizedText	Data rate for domain data transferred
Variable 2	Structure	
Name	String	FailureDetails
dataType	NodeId	StringNodeId
description	LocalizedText	Description of reason for abort

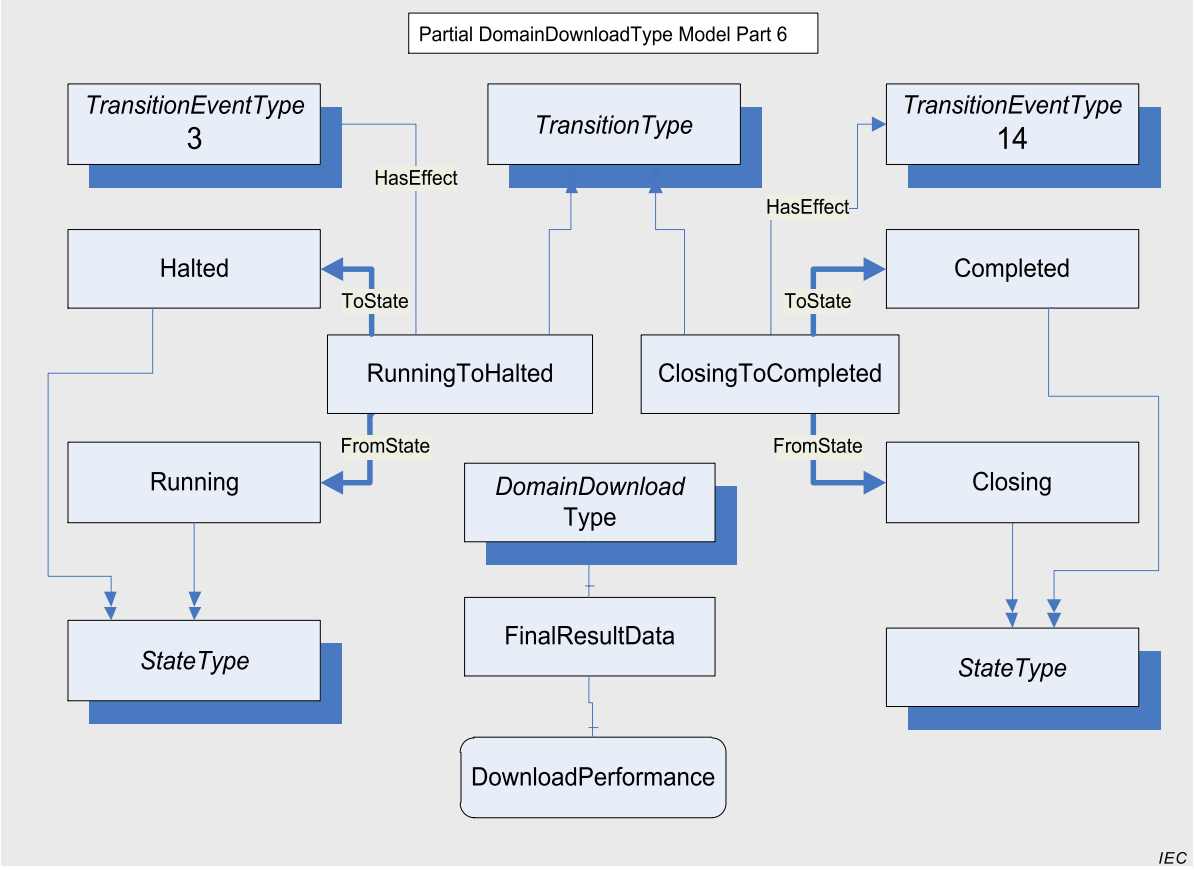

Figure A.10 – Running To Completed model

A.2.6.3 Sequence of operations

Figure A.11 illustrates a normal sequence of service exchanges between a *Client* and *Server* that would occur during the life cycle of a DomainDownloadType *Program Invocation*.

Copyright International Electrotechnical Commission

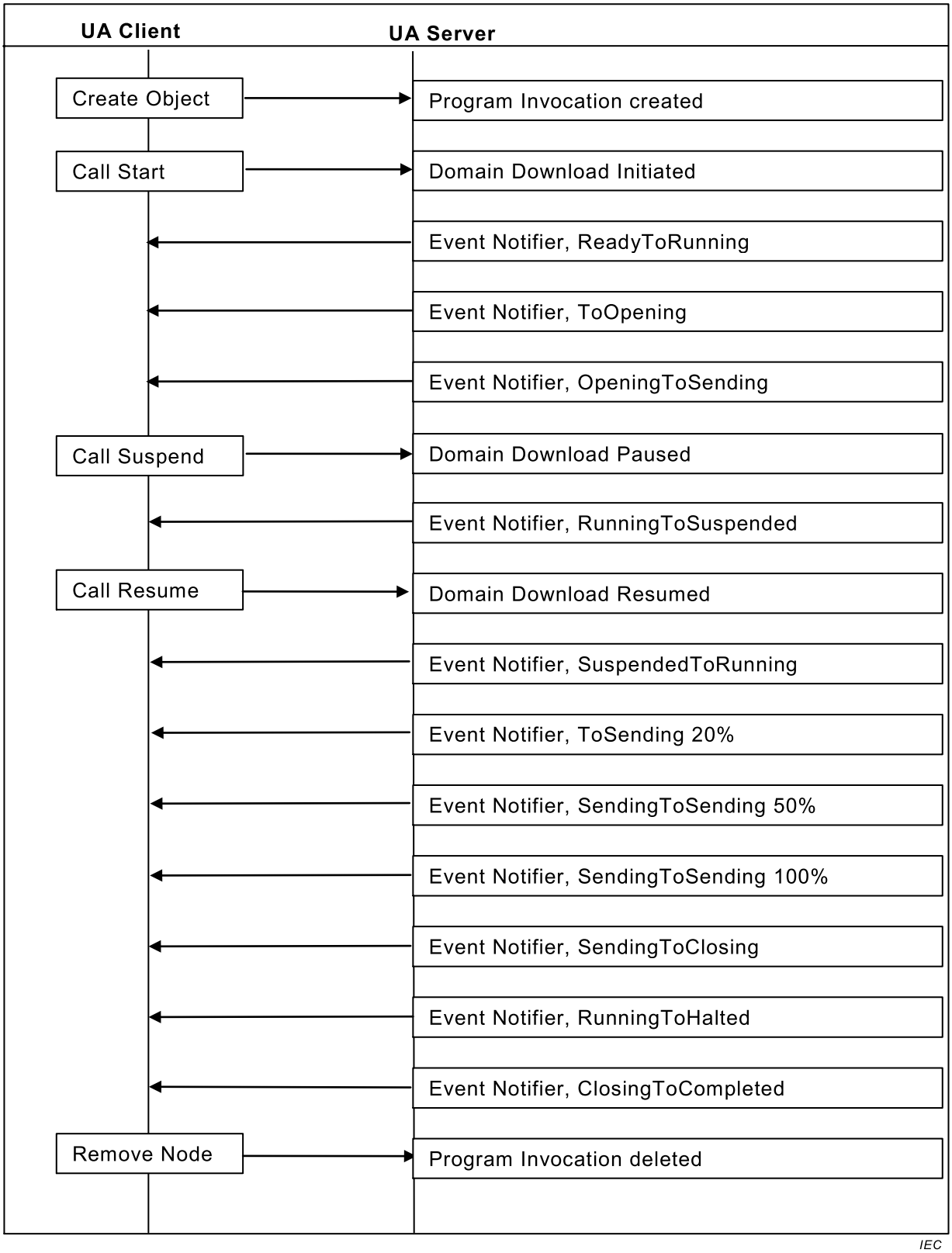

Figure A.11 – Sequence of operations

Copyright International Electrotechnical Commission

Copyright International Electrotechnical Commission

SOMMAIRE

Copyright International Electrotechnical Commission

Copyright International Electrotechnical Commission

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 10: Programmes

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-10 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2012. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

a) D'après l'examen du NIST, les considérations relatives à la sécurité ont été incluses en 4.2.2.

Copyright International Electrotechnical Commission

b) Transposition de la définition du Program Diagnostic Type dans un type de données (5.2.8) et ajout du type de données manquant pour la Program Diagnostic Variable dans le Type de Programme du Tableau 5.

c) Correction du Nom de Navigation des événements d'audit pour les Transitions de Programme dans le Tableau 7.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/383/FDIS	65E/409/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Architecture unifiée OPC*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

Copyright International Electrotechnical Commission

ARCHITECTURE UNIFIÉE OPC –

Partie 10: Programmes

1 Domaine d'application

La présente partie de l'IEC 62541 fait partie de la série de normes d'Architecture unifiée OPC (OPC UA) globale et définit le modèle d'informations associé aux *Programmes*. Elle comprend la description des *Classes de Nœuds,* et des *Propriétés, Méthodes* et *Événements* normalisés, ainsi que les paramètres associés relatifs au comportement et aux informations applicables aux *Programmes*.

Le modèle d'espace d'adresses complet, comprenant toutes les *Classes de Nœuds* et tous les *Attributs,* est spécifié dans l'IEC 62541-3. Les services tels que ceux utilisés pour invoquer les *Méthodes* appliquées pour gérer les *Programmes* sont spécifiés dans l'IEC 62541-4.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts* (disponible en anglais seulement)

IEC 62541-3:2015, *Architecture unifiée OPC – Partie 3: Modèle d'espace d'adresses*

IEC 62541-4:2015, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-5:2015, *Architecture unifiée OPC – Partie 5: Modèle d'Informations*

IEC 62541-7, *Architecture unifiée OPC – Partie 7: Profils*

3 Termes, définitions et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1 et l'IEC 62541-3, ainsi que les suivants s'appliquent.

3.1.1
fonction
function
tâche de programmation assurée par un serveur ou un dispositif, généralement réalisée par l'exécution de code machine

3.1.2
Diagramme d'Etats Finis
Finite State Machine
séquence d'états et de transitions d'états valides associés aux causes et effets correspondants définissant les actions d'un *Programme* en termes de stades discrets

Copyright International Electrotechnical Commission

3.1.3
Type de Programme
ProgramType
définition de type de Programme et qui constitue un sous-type du *FiniteStateMachineType* (Type de Diagramme d'Etats Finis)

3.1.4
Méthode de Commande de Programme
Program Control Method
méthode conçue selon la sémantique particulière de commande d'un *Programme* déclenchant une transition d'états

3.1.5
Invocation de Programme
Program Invocation
instance unique d'*Objet* d'un *Programme* existant sur un *Serveur*

Note 1 à l'article: L'*Invocation de Programme* se différencie des autres instances d'*Objet* du même *Type de Programme* par le chemin de navigation unique du nœud d'objet.

3.2 Abréviations

DA Data Access (Accès aux Données)

FSM Finite State Machine (Diagramme d'Etats Finis)

IHM Interfaces Homme-Machine

PCM Program Control Method (Méthode de Commande de Programme)

PGM Programme

PI Program Invocation (Invocation de Programme)

UA Unified Architecture (Architecture unifiée)

4 Concepts

4.1 Généralités

Les installations d'automatisation intégrée gèrent leurs opérations par des échanges de données et l'invocation coordonnée des fonctions système, tel qu'illustré à la Figure 1. Les *Services* sont nécessaires pour réaliser les échanges de données et invoquer les fonctions nécessaires au fonctionnement du système. Ces fonctions peuvent être appelées au moyen des Interfaces Homme-Machine, des contrôleurs cellulaires ou d'autres systèmes de commande de surveillance et d'acquisition de données. OPC UA définit les *Méthodes* et les *Programmes* comme des outils d'interopérabilité permettant de notifier, de découvrir et d'interroger ces fonctions. Ils fournissent un mécanisme de normalisation applicable à la description sémantique, à l'invocation et à la consignation des résultats de ces fonctions. Les *Méthodes* et les *Programmes* complètent les autres *Services* et *ObjectTypes (Types d'Objets)* d'OPC UA afin de faciliter l'exploitation d'un environnement automatisé basé sur une hiérarchie client-serveur.

Copyright International Electrotechnical Commission

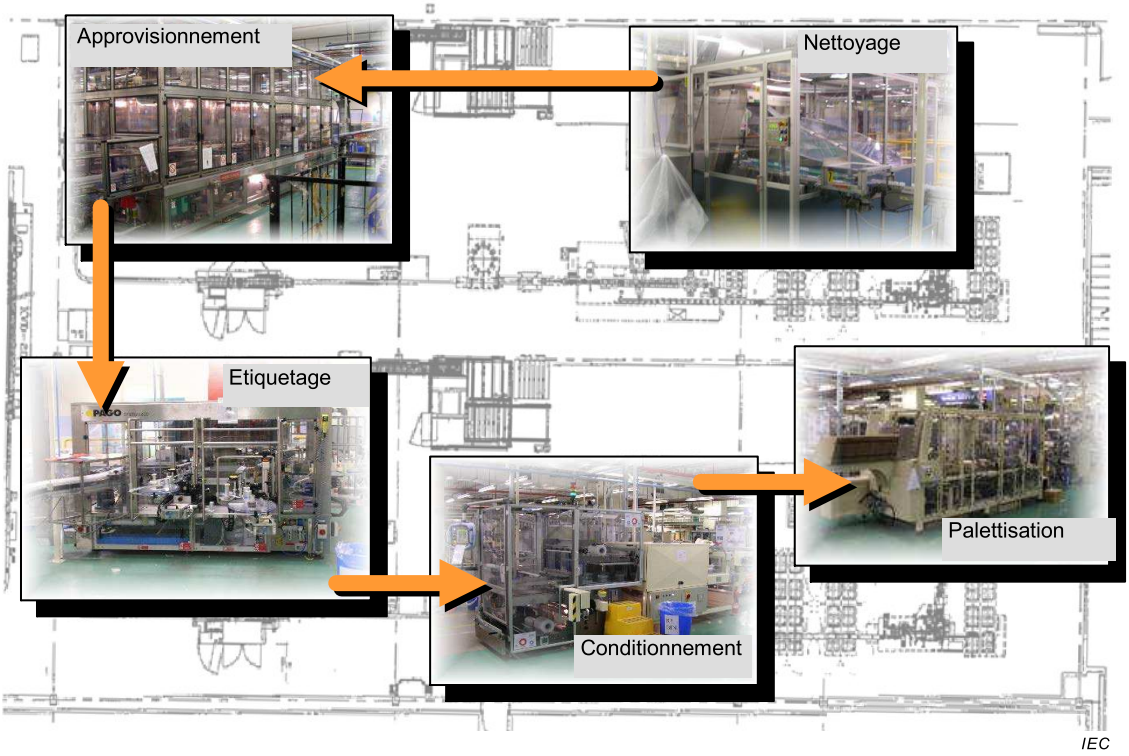

Figure 1 – Commande d'installation d'automatisation

Les fonctions du modèle de *Méthodes* et *Programmes* mises en œuvre dans les *Serveurs OPC* et les systèmes sous-jacents sont généralement régies par des champs d'application, des comportements, des durées de vie et des niveaux de complexité différents. En règle générale, ces fonctions ne sont pas basées sur des opérations de lecture ou d'écriture de données réalisées avec le jeu de service *Attribut* d'OPC UA.

Les *Méthodes* représentent les fonctions de base du *Serveur* que le *Client* peut invoquer. En revanche, les *Programmes* pour leur part permettent de modéliser une fonctionnalité dynamique plus complexe dans le système. Par exemple, un appel de Méthode peut être utilisé pour réaliser un calcul ou réinitialiser un compteur. Un *Programme* est utilisé pour exécuter et commander un traitement par lots, exécuter un programme-pièce de machine-outil ou gérer un téléchargement de domaine. Les *Méthodes* et leur mécanisme d'invocation sont décrits dans l'IEC 62541-3 et l'IEC 62541-4.

La présente norme décrit les extensions aux capacités fondamentales ou leur utilisation particulière requises pour les *Programmes*, telles que définies dans l'IEC 62541-5.

4.2 Programmes

4.2.1 Vue d'ensemble

Les *Programmes* sont des fonctions complexes résidant dans un serveur ou un système sous-jacent qu'un *Client* peut appeler et gérer. Les *Programmes* peuvent représenter n'importe quel niveau de fonctionnalité au sein d'un système ou d'un processus dont le déroulement est à contrôler et qui nécessite la commande ou l'intervention du client. La Figure 2 illustre le modèle.

Copyright International Electrotechnical Commission

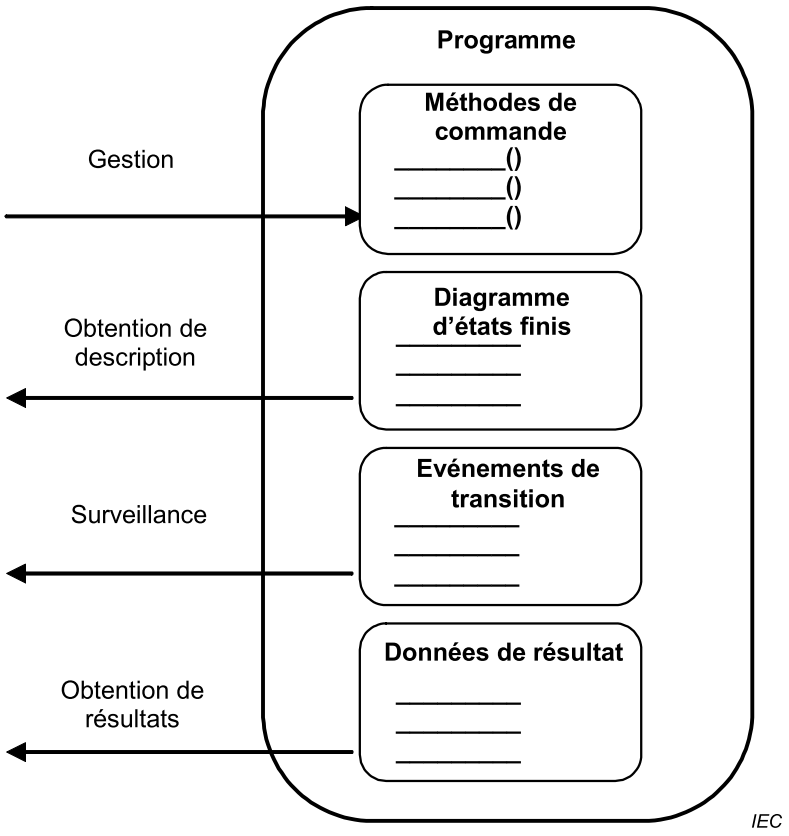

Figure 2 – Illustration d'un Programme

L'exécution des *Programmes* se déroule par transition selon une séquence définie d'états. Le comportement des *Programmes* est défini par un *Diagramme d'États Finis de Programme* (*PFSM)*. Les éléments du PFSM décrivent les phases d'exécution du *Programme* en termes de transitions valides qui s'exécutent entre les états, les stimuli ou causes des transitions considérées et les effets qui résultent des transitions.

4.2.2 Considérations relatives à la sécurité

Dans la mesure où les *Programmes* peuvent être utilisés pour réaliser des algorithmes de commande avancés ou autres actions, il convient de limiter leur utilisation au personnel disposant des droits d'accès appropriés. Il est recommandé que les AuditUpdateMethodEvents soient générés pour pouvoir surveiller le nombre de *Programmes* en cours d'exécution en plus de leur fréquence d'exécution.

4.2.3 Diagramme d'Etats Finis de Programme

Le Tableau 1 répertorie les états, transitions, causes et effets constitutifs du *Diagramme d'États Finis de Programme q*ui est illustré à la Figure 3.

Copyright International Electrotechnical Commission

Tableau 1 – Diagramme d'Etats Finis de Programme

N°	Dénomination de transition	Cause	De l'état	À l'état	Effet
1	HaltedToReady	Méthode Reset	Halted	Ready	Déclenche la Transition 1 Événement/Résultat
2	ReadyToRunning	Méthode Start	Ready	Running	Déclenche la Transition 2 Événement/Résultat
3	RunningToHalted	Méthode Halt ou Interne (erreur)	Running	Halted	Déclenche la Transition 3 Événement/Résultat
4	RunningToReady	Interne	Running	Ready	Déclenche la Transition 4 Événement/Résultat
5	RunningToSuspended	Méthode Suspend	Running	Suspended	Déclenche la Transition 5 Événement/Résultat
6	SuspendedToRunning	Méthode Resume	Suspended	Running	Déclenche la Transition 6 Événement/Résultat
7	SuspendedToHalted	Méthode Halt	Suspended	Halted	Déclenche la Transition 7 Événement/Résultat
8	SuspendedToReady	Interne	Suspended	Ready	Déclenche la Transition 8 Événement/Résultat
9	ReadyToHalted	Méthode Halt	Ready	Halted	Déclenche la Transition 9 Événement/Résultat

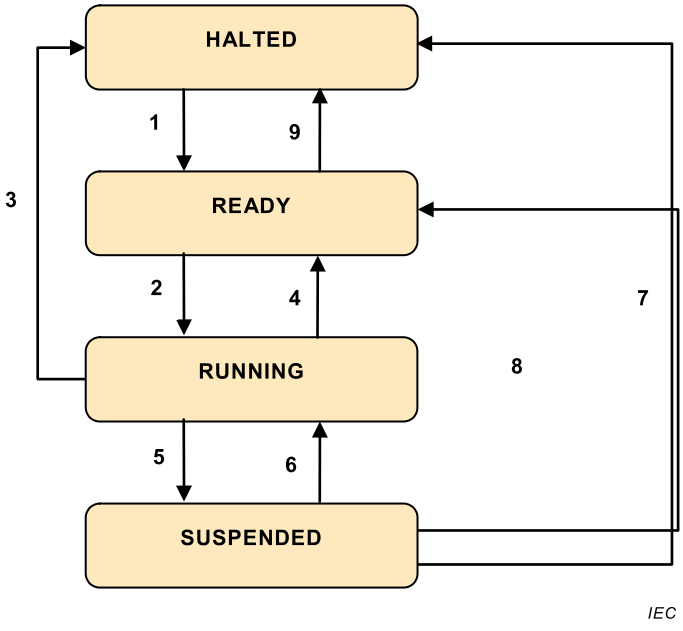

Figure 3 – États et transitions d'un Programme

4.2.4 États d'un Programme

Un ensemble normalisé d'états de base est défini pour les *Programmes* faisant partie intégrante du *Diagramme d'États Finis de Programme.* Ces états représentent les stades auxquels un *Programme* peut exister à un moment donné tel que perçu par un *Client*. Cet état est considéré comme l'état courant du *Programme*. Tous les *Programmes* doivent prendre en charge cet ensemble de base. Un *Programme* peut ou non nécessiter une action du *Client* pour voir son état modifié. Le Tableau 2 définit de manière formelle les états.

Copyright International Electrotechnical Commission

Tableau 2 – États d'un Programme

Etat	Description
Ready	Le *Programme* est correctement initialisé et peut être démarré.
Running	Le *Programme* est en cours d'exécution jusqu'à son achèvement.
Suspended	Le *Programme* a été interrompu avant d'atteindre un état final mais il peut être repris.
Halted	Le *Programme* est parvenu à un état final ou d'échec (failed); il ne peut pas être démarré ou repris sans réinitialisation.

L'ensemble des états définis pour décrire un *Programme* peut être étendu. Les sous-états de *Programme* peuvent être définis de sorte que les états de base fournissent une plus grande résolution d'un processus et décrivent la cause et le ou les effets des stimuli et transitions supplémentaires. Les organismes de normalisation et les groupes industriels peuvent étendre le *Modèle d'États Finis de Programme* de base pour se conformer aux différents modèles industriels. Par exemple, l'état Halted (arrêté) peut comprendre les sous-états "Aborted" (abandonné) et "Completed" (fini) permettant d'indiquer si la fonction a été achevée de manière satisfaisante avant de passer à la transition à l'état Halted. Les états de transition tels que "Starting" (démarrage) ou "Suspending" (interruption) peuvent également être des extensions de l'état Running (en cours d'exécution), par exemple.

4.2.5 Transitions d'états

Un ensemble normalisé de transitions d'états est défini pour le *Diagramme d'États Finis de Programme.* Ces transitions définissent les modifications valides apportées à l'état courant du *Programme* en termes d'état initial et d'état résultant. Le Tableau 3 définit de manière formelle les transitions.

Tableau 3 – Transitions d'états d'un Programme

N° de transition	Dénomination de transition	État initial	État résultant
1	HaltedToReady	Halted	Ready
2	ReadyToRunning	Ready	Running
3	RunningToHalted	Running	Halted
4	RunningToReady	Running	Ready
5	RunningToSuspended	Running	Suspended
6	SuspendedToRunning	Suspended	Running
7	SuspendedToHalted	Suspended	Halted
8	SuspendedToReady	Suspended	Ready
9	ReadyToHalted	Ready	Halted

4.2.6 Stimuli de transition d'état d'un Programme

Les stimuli ou les causes de transitions d'états d'un *Programme* peuvent être internes ou externes au *Serveur*. Les stimuli internes sont par exemple la réalisation complète des étapes d'usinage, la détection d'une condition d'alarme ou la transmission d'un paquet de données. Les stimuli externes sont par exemple les *Méthodes*. Les *Méthodes n*ormalisées sont définies pour agir comme des stimuli pour la commande d'un *Programme*.

4.2.7 Méthodes de Commande de Programme

Les *Clients* gèrent un *Programme* en appelant les *Méthodes.* Les *Méthodes* agissent sur le comportement d'un *Programme* en provoquant les transitions d'états spécifiées. Les transitions d'états régissent les actions entreprises par le *Programme*. La présente norme définit un ensemble de *Méthodes de Commande de Programme* normalisées. Ces *Méthodes* fournissent un nombre suffisant d'informations au client pour exécuter un *Programme*.

Copyright International Electrotechnical Commission

Le Tableau 4 répertorie l'ensemble des *Méthodes de Commande de Programme* définies. Chaque *Méthode* provoque des transitions à partir d'états spécifiés, elle doit être appelée lorsque le *Programme* est parvenu à l'un des états spécifiés.

Des *Programmes* individuels peuvent également prendre en charge n'importe quel sous-ensemble des *Méthodes de Commande de Programme.* Par exemple, certains *Programmes* peuvent ne pas être autorisés à déclencher un état d'interruption et ne fournissent de ce fait pas les *Méthodes Suspend* et *Resume.*

Les *Programmes* peuvent prendre en charge des *Méthodes* supplémentaires définies par l'utilisateur. Les *Méthodes* définies par l'utilisateur ne doivent pas modifier le comportement du *Diagramme d'États Finis de Programme* de base.

Tableau 4 – Méthodes de Commande de Programme

Dénomination de méthode	Description
Start	Fait passer le *Programme* de l'état Ready à l'état Running.
Suspend	Fait passer le *Programme* de l'état Running à l'état Suspended.
Resume	Fait passer le *Programme* de l'état Suspended à l'état Running.
Halt	Fait passer le *Programme* de l'état Ready, Running ou Suspended à l'état Halted.
Reset	Fait passer le *Programme* de l'état Halted à l'état Ready.

Les *Méthodes de Commande d'un Programme* peuvent inclure des arguments qui sont utilisés par le *Programme.* Par exemple, une *Méthode* Start (démarrage) peut inclure un argument d'options qui spécifie des options dynamiques utilisées pour déterminer un comportement donné du programme. Les arguments peuvent être différents pour chaque *Type de Programme*. Le service Appel de Méthode spécifié à l'IEC 62541-4:2015, 5.11, définit un état de retour. Cet état de retour indique la réussite de la *Méthode de Commande de Programme* ou la raison de son échec.

4.2.8 Effets des transitions d'états d'un Programme

Une transition d'états d'un *Programme* a généralement une cause et produit également un effet. L'effet est un sous-produit d'une transition d'états d'un *Programme* qu'un *Client* peut utiliser pour surveiller l'avancement du *Programme*. Les effets peuvent être internes ou externes. Un effet externe d'une transition d'états est caractérisé par la génération d'une notification d'*Evénement*. Chaque transition d'états d'un *Programme* est associée à un *Evénement* unique. Ces *Evénements* reflètent la progression et la trajectoire du *Programme* dans son ensemble d'états définis. Les effets internes d'une transition d'états peuvent se caractériser par la réalisation d'une action de programmation donnée telle que la génération de données.

4.2.9 Données de résultat d'un Programme

4.2.9.1 Vue d'ensemble

Les données de résultat sont générées par un *Programme* Running (en cours d'exécution). Les données de résultat peuvent être intermédiaires ou finales. Les données de résultat peuvent être associées à des transitions d'états particulières du *Programme*.

4.2.9.2 Données intermédiaires de résultat

Les données intermédiaires de résultat sont transitoires et générées par le *Programme* conjointement aux transitions d'états non finales. Les éléments de données constitutifs des résultats intermédiaires sont définis en association avec les transitions d'états particulières du *Programme*. Leurs valeurs ne sont significatives qu'au niveau de la transition.

Copyright International Electrotechnical Commission

Chaque transition d'états d'un *Programme* peut être associée à différents éléments de données de résultat. En variante, un ensemble de transitions peut partager un élément de données de résultat. Le pourcentage complet est un exemple de données intermédiaires de résultat. La valeur du pourcentage complet est obtenue lorsque la transition d'états se produit et est disponible pour le *Client*.

Les *Clients* acquièrent les données intermédiaires de résultat par abonnement aux *Evénements* de transition d'états d'un *Programme*. Les *Evénements* spécifient les éléments de données pour chaque transition. Lorsque la transition se produit, l'*Evénement* généré transmet les valeurs des données de résultat obtenues aux *Clients* abonnés. Si aucun *Client* ne surveille le *Programme*, les données intermédiaires de résultat peuvent être omises.

4.2.9.3 Données finales de résultat

Les données finales de résultat sont générées à la fin de l'exécution du *Programme*. Le temps d'exécution total, le nombre d'objets fenêtre et une condition de défaut sont des exemples de données finales de résultat. Lorsque le *Programme* passe à l'état final, ces données de résultat peuvent être transmises au client par l'*Evénement* de transition. Les données finales de résultat disponibles dans le *Programme* peuvent également être lues par un *Client* après l'arrêt du programme. Ces données restent présentes jusqu'à nouvelle exécution ou suppression de l'Instance d'un *Programme.*

4.2.9.4 Surveillance des Programmes

Les *Clients* peuvent surveiller les activités associées à l'exécution du *Programme*. Ces activités comprennent l'invocation des *Méthodes* de gestion, la génération de données de résultat et la progression du *Programme* dans ses états. Les événements d'Audit sont fournis par les *Appels de Méthode* et les transitions d'états. Ces *Evénements* permettent de conserver l'enregistrement des *Clients* qui interagissent avec un *Programme* et les transitions d'états du *Programme* résultants de cette interaction.

4.2.10 Durée de vie d'un Programme

4.2.10.1 Vue d'ensemble

Les *Programmes* peuvent avoir différentes durées de vie. Certains *Programmes* peuvent résider de manière permanente sur un *Serveur* alors que d'autres sont créés et retirés. La création et le retrait peuvent être contrôlés par un *Client* ou être limités à un dispositif local.

Un *Programme* peut être créé par le *Client*. Dans ce cas, le *Client* peut ajouter le *Programme* au *Serveur*. La *Méthode de Création d'Objet* définie dans l'IEC 62541-3:2015, 5.5.4, permet de créer l'instance d'un *Programme*. L'état initial du *Programme* peut être *Halted* (*Arrêté*) ou *Ready* (*Prêt*). Certains *Programmes*, par exemple, peuvent nécessiter de disposer d'une ressource après leur création, avant de pouvoir être à l'état prêt pour l'exécution. Dans ce cas, il serait initialisé à l'état *Halted* pour ensuite passer à l'état Ready lorsque la ressource est fournie.

Un Programme peut être retiré par le *Client*. Dans ce cas, le *Client* peut supprimer l'instance du *Programme* du *Serveur*. Le *Service DeleteNodes (Suppression de Nœuds)* défini dans l'IEC 62541-4 permet de retirer l'Instance d'un *Programme*. Le *Programme* doit être à l'état *Halted* pour être retiré. Un *Programme* peut également être à retrait automatique. Dans ce cas, le *Programme* se supprime lui-même à la fin de son exécution.

4.2.10.2 Instances d'un Programme

Les *Programmes* peuvent avoir une seule ou plusieurs instances. Un *Serveur* peut prendre en charge plusieurs instances d'un *Programme* si elles peuvent être exécutées en parallèle. Par exemple, le *Programme* peut définir une *Méthode Start* qui a un argument d'entrée destiné à spécifier la ressource qui est exécutée par ses fonctions. Chaque instance du *Programme* est ensuite démarrée en désignant l'utilisation des différentes ressources. Le *Client* peut

Copyright International Electrotechnical Commission

découvrir toutes les instances d'un *Programme* en cours d'exécution sur un *Serveur*. Chaque Instance d'un *Programme* est identifiée de manière unique sur le *Serveur* et est gérée de manière indépendante par le *Client*.

4.2.10.3 Cycle de vie d'un programme

Les *Programmes* peuvent être exécutés une fois ou plusieurs fois (cycliques). Un *Programme* exécuté une fois reste toujours à l'état *Halted* après son exécution. Le déroulement normal de l'exécution serait de le supprimer après vérification des résultats finaux.

Les *Programmes* cycliques peuvent comporter un nombre de cycles limité ou illimité. Ces *Programmes* peuvent nécessiter de procéder à une étape de réinitialisation pour passer de l'état *Halted* à l'état *Ready*. Ceci permet de reconstituer les ressources ou de réinitialiser les paramètres avant de redémarrer le *Programme*. La *Méthode de Commande d'un Programme* "*Reset*" (*Réinitialisation*) déclenche cette transition d'états et toute action ou tout effet associé(e).

5 Modèle

5.1 Généralités

Le modèle d'un *Programme* étend le modèle *FiniteStateMachineType* (*Type de Diagramme d'États Finis*) et le modèle de base *ObjectType* (*Type d'Objet*) présentés dans l'IEC 62541-5. Chaque *Programme* comporte une *Définition de Type* qui constitue le sous-type du *FiniteStateMachineType.* Le *Type de Programme* décrit le modèle *Diagramme d'Etats Finis* pris en charge par toute *Invocation de Programme* du type considéré. Le *Type de Programme* définit également le jeu de propriétés qui caractérise les aspects particuliers du comportement du *Programme* tels que la durée de vie et le cycle de fonctionnement, ainsi que les données de résultat générées par le *Programme*.

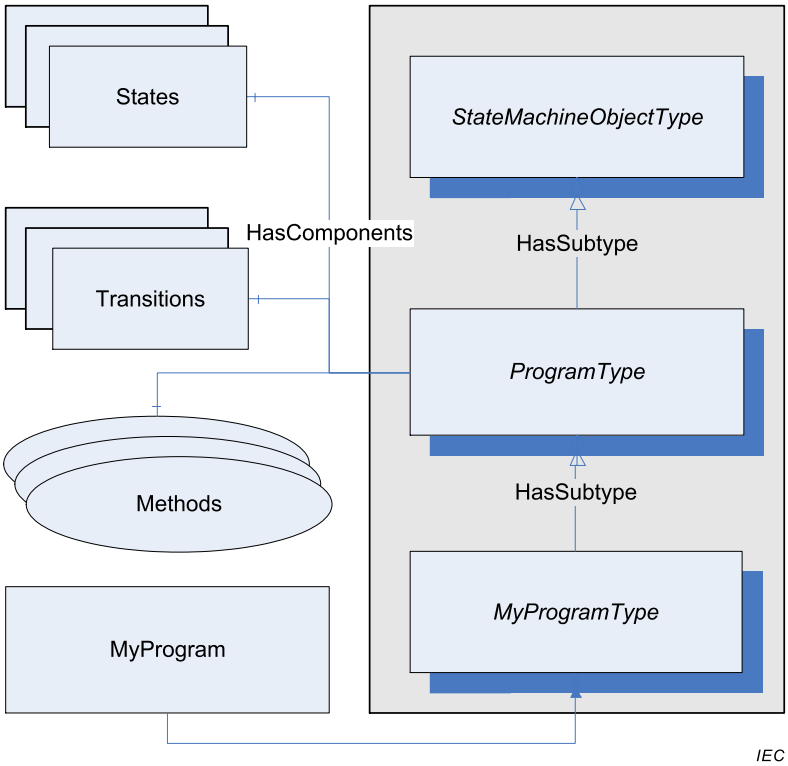

Figure 4 – Type de Programme

Copyright International Electrotechnical Commission

Le *Type de Programme* de base définit le *Diagramme d'Etats Finis* normalisé spécifié pour tous les *Programmes*. Ceci comprend les états, les transitions et les causes (*Méthodes*) et les effets (*Événements*) des transitions. Les sous-types du *Type de Programme* de base peuvent être définis pour étendre ou caractériser plus spécifiquement le comportement d'un *Programme* individuel, comme illustré par le "MyProgramType" de la Figure 4.

5.2 Type de Programme

5.2.1 Vue d'ensemble

Le Tableau 5 énumère les propriétés et les composants supplémentaires qui constituent le *Type de Programme*. Aucune sémantique spécifique au *Type de Programme* n'est attribuée aux autres *Attributs* ou *Propriétés* de base de *ObjectType* ou *FiniteStateMachineType*.

Tableau 5 – Type de Programme

Attribut	**Valeur**				
	Comprend tous les attributs spécifiés pour le FiniteStateMachineType				
BrowseName	Type de Programme				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasProperty	Variable	Creatable	Boolean	PropertyType	--
HasProperty	Variable	Deletabe	Boolean	PropertyType	Obligatoire
HasProperty	Variable	AutoDelete	Boolean	PropertyType	Obligatoire
HasProperty	Variable	RecycleCount	Int32	PropertyType	Obligatoire
HasProperty	Variable	InstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxInstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxRecycleCount	UInt32	PropertyType	--
HasComponent	Variable	ProgramDiagnostic	ProgramDiagnosticDataType	ProgramDiagnosticType	Facultative
HasComponent	Objet	Halted		StateType	Obligatoire
HasComponent	Objet	Ready		StateType	Obligatoire
HasComponent	Objet	Running		StateType	Obligatoire
HasComponent	Objet	Suspended		StateType	Obligatoire
HasComponent	Objet	HaltedToReady		TransitionType	Obligatoire
HasComponent	Objet	ReadyToRunning		TransitionType	Obligatoire
HasComponent	Objet	RunningToHalted		TransitionType	Obligatoire
HasComponent	Objet	RunningToReady		TransitionType	Obligatoire
HasComponent	Objet	RunningToSuspended		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToRunning		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToHalted		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToReady		TransitionType	Obligatoire
HasComponent	Objet	ReadyToHalted		TransitionType	Obligatoire

Copyright International Electrotechnical Commission

Attribut	Valeur				
	Comprend tous les attributs spécifiés pour le FiniteStateMachineType				
BrowseName	Type de Programme				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasComponent	Méthode	Start			Facultative
HasComponent	Méthode	Suspend			Facultative
HasComponent	Méthode	Reset			Facultative
HasComponent	Méthode	Halt			Facultative
HasComponent	Méthode	Resume			Facultative
HasComponent	Objet	FinalResultData		BaseObjectType	Facultative

5.2.2 Propriétés de Type de Programme

La *Propriété Creatable* (*Créable*) est un booléen qui indique si les *Invocations de Programme* de ce *Type de Programme* peuvent être créées par un *Client*. Si la valeur est Faux, ces *Invocations de Programme* sont persistantes ou ne peuvent être créées que par le *Serveur*.

La *Propriété Deletable* (*Effaçable*) est un booléen qui indique si une *Invocation de Programme* de ce *Type de Programme* peut être supprimée par un *Client*. Si la valeur est Faux, ces *Invocations de Programme* ne peuvent être supprimées que par le *Serveur*.

La *Propriété AutoDelete* (*Suppression Automatique*) est un booléen qui indique si les *Invocations de Programme* de ce *Type de Programme* sont retirées par le *Serveur* à la fin de l'exécution. Si la valeur est Faux, ces *Invocations de Programme* demeurent sur le *Serveur* jusqu'à leur suppression par le *Client*. Lorsque *l'Invocation de Programme* est supprimée, toutes données de résultat associées à l'instance sont également retirées.

La *Propriété RecycleCount* (*Nombre de Cycles*) est un entier non signé qui indique le nombre de fois qu'une *Invocation de Programme* de ce type a exécuté un cycle ou a été redémarrée depuis son démarrage (et non reprise). À noter que la Méthode Reset peut être appliquée pour préparer le redémarrage d'un *Programme*.

La *Propriété MaxRecycleCount* (*Nombre Maximal de Cycles*) est un entier qui indique le nombre maximal de fois qu'une *Invocation de Programme* de ce type peut exécuter un cycle ou être redémarrée depuis son démarrage (et non reprise). Si la valeur est inférieure à 0, le nombre de redémarrages est illimité. Si la valeur est égale à zéro, le Programme ne peut exécuter de cycle ni être redémarré.

La *Propriété InstanceCount* (*Nombre d'Instances*) est un entier non signé qui indique le nombre d'*Invocations de Programme* de ce type qui existent réellement.

La *Propriété MaxInstanceCount* (*Nombre Maximal d'Instances*) est un entier qui indique le nombre maximal d'*Invocations de Programme* de ce type qui peuvent exister simultanément sur ce *Serveur*. Si la valeur est inférieure à 0, le nombre est illimité.

Copyright International Electrotechnical Commission

5.2.3 Composants de Type de Programme

5.2.3.1 Vue d'ensemble

Les composants de *Type de Programme* comprennent un jeu de *Références* aux instances d'*Objet* de *StateTypes* (*Types d'Etats*), *TransitionTypes* (*Types de Transitions*), *EventTypes* (*Types d'Événements*) et les *Méthodes* qui dans leur ensemble définissent le *FiniteStateMachine* du *Programme.*

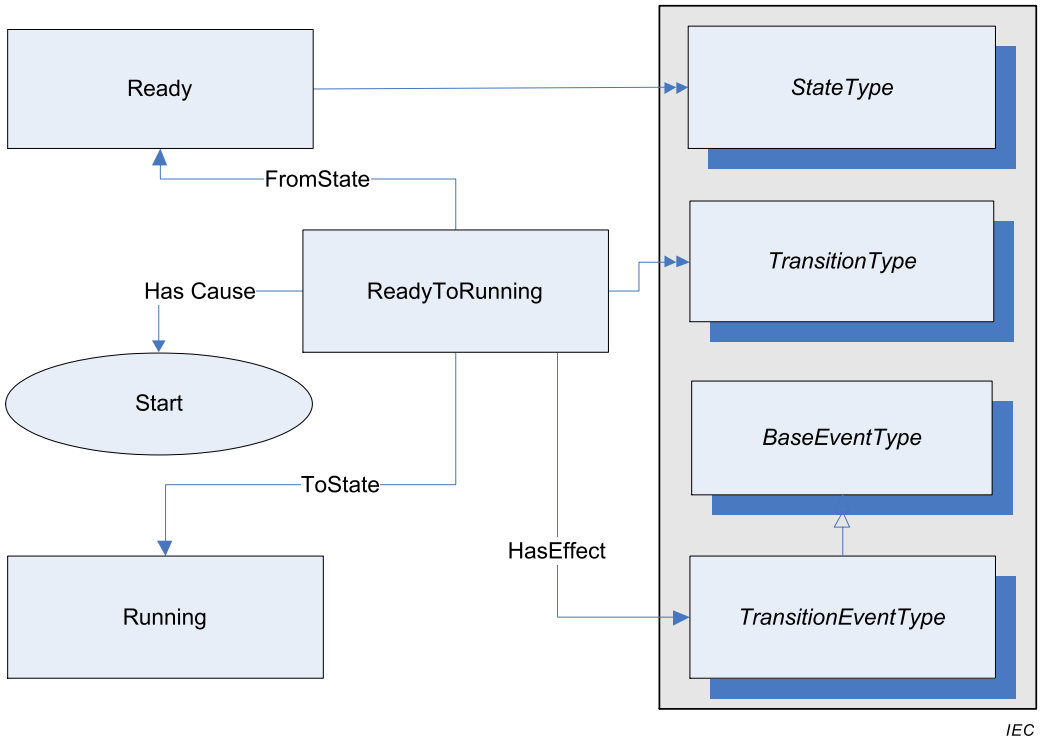

Figure 5 – Références FSM d'un Programme

La Figure 5 illustre les *Références* de composant qui définissent les associations entre les deux états de *Type de Programme*, *Ready* et *Running*. Les ReferenceTypes (Types de Références) supplémentaires ont été omis pour simplifier la représentation.

5.2.3.2 États Type de Programme

Le Tableau 6 spécifie les *Objets* d'état *Type de Programme*. Ces *Objets* sont les instances du *StateType* (Type d'État) définies dans l'IEC 62541-5:2015, Annexe B . Une valeur unique *StateNumber* est attribuée à chaque état. Les sous-types du *Type de Programme* peuvent ajouter des références de n'importe quel état à un *Objet StateMachine* subordonné ou imbriqué pour étendre le *FiniteStateMachine*.

Copyright International Electrotechnical Commission

Tableau 6 – États d'un Programme

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	Notes
États					
Halted	HasProperty	StateNumber	1	PropertyType	
	ToTransition	HaltedToReady		TransitionType	
	FromTransition	RunningToHalted		TransitionType	
	FromTransition	SuspendedToHalted		TransitionType	
	FromTransition	ReadyToHalted		TransitionType	
Ready	HasProperty	StateNumber	2	PropertyType	
	FromTransition	HaltedToReady		TransitionType	
	ToTransition	ReadyToRunning		TransitionType	
	FromTransition	RunningToReady		TransitionType	
	ToTransition	ReadyToHalted		TransitionType	
Running	HasProperty	StateNumber	3	PropertyType	
	ToTransition	RunningToHalted		TransitionType	
	ToTransition	RunningToReady		TransitionType	
	ToTransition	RunningToSuspended		TransitionType	
	FromTransition	ReadyToRunning		TransitionType	
	FromTransition	SuspendedToRunning		TransitionType	
Suspended	HasProperty	StateNumber	4	PropertyType	
	ToTransition	SuspendedToRunning		TransitionType	
	ToTransition	SuspendedToHalted		TransitionType	
	ToTransition	SuspendedToReady		TransitionType	
	FromTransition	RunningToSuspended		TransitionType	

L'état *Halted* (*Arrêté*) est l'état au repos d'un *Programme*. Il peut s'agir d'un état initial ou d'un état final. S'agissant d'un état initial, l'*Invocation de Programme n*e peut pas commencer l'exécution du fait des conditions existant sur le *Serveur*. S'agissant d'un état final, l'état *Halted* peut indiquer un état "failed" (échec) ou un état "completed" (fini) du *Programme*. Un état ou résultat subordonné peut être utilisé pour déterminer la nature de l'achèvement du programme. L'état *Halted* fait référence à quatre *Objets de Transition*, qui identifient les transitions d'états autorisées vers l'état *Ready* et depuis les états *Ready*, *Running* et *Suspended*.

L'état *Ready* (*Prêt*) indique que le *Programme* est prêt à commencer l'exécution. Les *Programmes* qui sont prêts à commencer l'exécution dès leur création peuvent immédiatement passer à l'état *Ready*. L'état *Ready* fait référence à quatre *Objets de Transition,* qui identifient les transitions d'états autorisées vers les états *Running* et *Halted* et depuis les états *Halted* et *Ready*.

L'état *Running* (*En cours*) indique que le *Programme* est en cours de réalisation de sa fonction. L'état *Running* fait référence à cinq *Objets de Transition,* qui identifient les transitions d'états autorisées vers les états *Halted*, *Ready* et *Suspended* et depuis les états *Ready* et *Suspended*.

Copyright International Electrotechnical Commission

L'état *Suspended* (*Interrompu*) indique que le *Programme* a arrêté la réalisation de sa fonction, mais conserve la capacité de reprendre la fonction au point auquel elle a été interrompue. L'état *Suspended* fait référence à quatre *Objets de Transition*, qui identifient les transitions d'états autorisées vers les états *Ready*, *Running* et *Halted* et depuis l'état *Ready*.

5.2.3.3 Transitions Type de Programme

Les Transitions *Type de Programme* sont les instances du *TransitionType* (*Type de Transition*) définies dans l'IEC 62541-5:2015, Article B.4, qui comprend également les définitions des références ToState, FromState, HasCause et HasEffect utilisées. Le Tableau 7 spécifie les transitions définies pour le *Type de Programme.* Un *TransitionNumber* unique est attribué à chaque transition. La colonne Notes indique lorsqu'une cause fait référence à des *Méthodes* et lorsque les effets sont facultatifs.

Tableau 7 – Transitions d'un Programme

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
Transitions					
HaltedToReady	HasProperty	TransitionNumber	1	PropertyType	
	ToState	Ready		StateType	
	FromState	Halted		StateType	
	HasCause	Reset			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToRunning	HasProperty	TransitionNumber	2	PropertyType	
	ToState	Running		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToHalted	HasProperty	TransitionNumber	3	PropertyType	
	ToState	Halted		StateType	
	FromState	Running		StateType	
	HasCause	Halt			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToReady	HasProperty	TransitionNumber	4	PropertyType	
	ToState	Ready		StateType	
	FromState	Runnning		StateType	
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
RunningToSuspended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	

Copyright International Electrotechnical Commission

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
Transitions					
	HasCause	Suspend			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToRunning	HasProperty	TransitionNumber	6	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Resume			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToHalted	HasProperty	TransitionNumber	7	PropertyType	
	ToState	Halted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToReady	HasProperty	TransitionNumber	8	PropertyType	
	ToState	Ready		StateType	
	FromState	Suspended		StateType	
	HasCause	Reset			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToHalted	HasProperty	TransitionNumber	9	PropertyType	
	ToState	Halted		StateType	
	FromState	Ready		StateType	
	HasCause	Halt			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			

La transition *HaltedToReady* spécifie la transition de l'état *Halted* à l'état *Ready*. Elle peut être déclenchée par la *Méthode Reset.*

La transition *ReadyToRunning* spécifie la transition de l'état *Ready* à l'état *Running*. Elle est déclenchée par la *Méthode Start.*

La transition *RunningToHalted* spécifie la transition de l'état *Running* à l'état *Halted*. Elle est déclenchée par la *Méthode Halt.*

La transition *RunningToReady* spécifie la transition de l'état *Running* à l'état *Ready*. La transition *RunningToSuspended* spécifie la Transition de l'état *Running* à l'état *Suspended*. Elle est déclenchée par la *Méthode Suspend.*

Copyright International Electrotechnical Commission

La transition *SuspendedToRunning* spécifie la transition de l'état *Suspended* à l'état *Running*. Elle est déclenchée par la *Méthode Resume.*

La transition *SuspendedToHalted* spécifie la transition de l'état *Suspended* à l'état *Halted*. Elle est déclenchée par la *Méthode Halt.*

La transition *SuspendedToReady* spécifie la transition de l'état *Suspended* à l'état *Ready*. Sa cause est interne.

La transition *ReadyToHalted* spécifie la transition de l'état *Ready* à l'état *Halted*. Elle est déclenchée par la *Méthode Halt.*

Deux références *HasEffect* sont spécifiées pour chaque transition du Programme. Ces effets sont des *Evénements* de *ProgramTransitionEventType* et *AuditProgramTransitionEventType* définis en 5.2.5. Le *ProgramTransitionEventType* informe les *Clients* de la transition d'un Programme et véhicule les données de résultat. Le *AuditProgramTransitionEventType* est utilisé pour l'audit des transitions résultant des *Méthodes de Commande du Programme*.

Le Profil "Audit Server Facet" défini dans l'IEC 62541-7 nécessite le support du *AuditProgramTransitionEventType.*

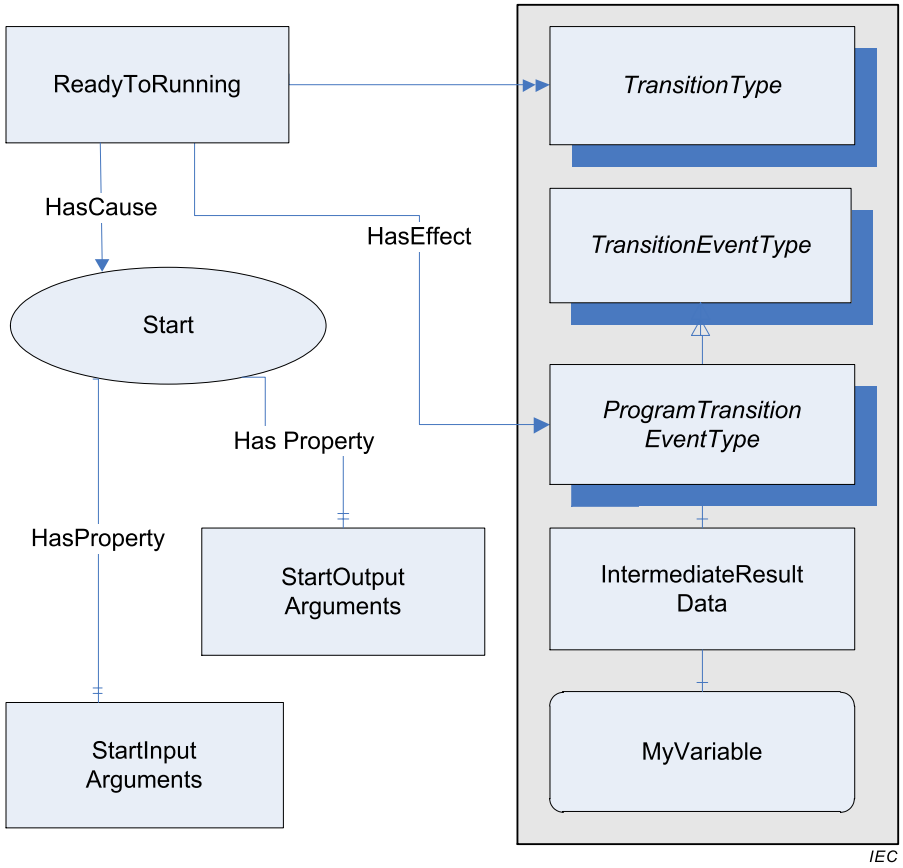

Figure 6 – Causes et effets Type de Programme

5.2.4 Causes Type de Programme (Méthodes)

5.2.4.1 Vue d'ensemble

Le *Type de Programme* comprend des références aux *Causes* des transitions d'états particulières du *Programme*. Ces causes font référence aux instances de *Méthode.* Les *Programmes* qui ne prennent pas en charge une *Méthode de Commande d'un Programme,* ne

Copyright International Electrotechnical Commission

tiennent pas compte de la référence *Causes* de cette *Méthode* issue des références de *Type de Programme*. Si une référence *Causes de Méthode* est omise du *Type de Programme*, le *Client* ne peut pas provoquer la transition d'états associée. Les instances de *Méthode* référencées par le *Type de Programme* identifient les InputArguments (Arguments d'Entrée) et les OutputArguments (Arguments de sortie) nécessaires pour les appels de *Méthode* aux *Invocations de Programme* du Type de Programme considéré. Le Tableau 8 spécifie les *Méthodes* définies comme *Causes* des *ProgramTypes (Types de Programmes)*. La Figure 6 illustre les *Références* qui associent les composants et les *Propriétés* des *Méthodes* et *Événements* avec les Program transitions (transitions d'un Programme).

Tableau 8 – Causes Type de Programme

Nom de navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
Causes					
Start	HasProperty	InputArguments		PropertyType	Facultative
	HasProperty	OutputArguments		PropertyType	Facultative
Suspend	HasProperty	InputArguments		PropertyType	Facultative
	HasProperty	OutputArguments		PropertyType	Facultative
Resume	HasProperty	InputArguments		PropertyType	Facultative
	HasProperty	OutputArguments		PropertyType	Facultative
Halt	HasProperty	InputArguments		PropertyType	Facultative
	HasProperty	OutputArguments		PropertyType	Facultative
Reset	HasProperty	InputArguments		PropertyType	Facultative
	HasProperty	OutputArguments		PropertyType	Facultative

La *Méthode Start* provoque la transition d'un *Programme ReadyToRunning*.

La *Méthode Suspend* provoque la transition d'un *Programme RunningToSuspended*.

La *Méthode Resume* provoque la transition d'un *Programme SuspendedToRunning*.

La *Méthode Halt* provoque la transition d'un *Programme RunningToHalted, SuspendedToHalted* ou *ReadyToHalted* en fonction de l'état courant) du *Programme*.

La *Méthode Reset* provoque la transition d'un *Programme HaltedToReady*.

5.2.4.2 Attributs normalisés

L'attribut *Méthode Executable* indique si une méthode peut effectivement être exécutée. Pour les *Méthodes de Commande du Programme*, ceci signifie que le *Programme* principal a un état courant qui prend en charge la transition provoquée par la *Méthode.*

5.2.4.3 Propriétés normalisées

5.2.4.3.1 Vue d'ensemble

Les *Méthodes* peuvent faire référence à un jeu d'*InputArguments* (Arguments d'Entrée). Pour chaque *Type de Programme*, un jeu d'*InputArguments* peut être défini pour les *Méthodes de*

Copyright International Electrotechnical Commission

Commande du Programme prises en charge. Les données transmises dans les arguments complètent les informations requises par le *Programme* pour réaliser sa fonction. Tous les appels à une *Méthode de Commande du Programme* pour chaque *Invocation de Programme* de ce *Type* doivent satisfaire aux arguments spécifiés.

Les *Méthodes* peuvent faire référence à un jeu d'*OutputArguments* (Arguments de Sortie). Pour chaque *Type de Programme*, un jeu d'*OutputArguments* est défini pour les *Méthodes de Commande du Programme* prises en charge. Tous les appels à une *Méthode de Commande du Programme* pour chaque *Invocation de Programme* de ce *Type* doivent satisfaire aux arguments spécifiés.

5.2.5 Effets Type de Programme (Événements)

5.2.5.1 Vue d'ensemble

Le *Type de Programme* comprend des références de composant aux *Effets* de chacune des transitions d'états du *Programme*. Ces *Effets* sont des *Événements.* Chaque *Transition* doit avoir une *Référence HasEffect* à un *ProgramTransitionEventType* et peut avoir un *AuditProgramTransitionEventType.* Lorsque la transition se produit, des notifications d'*Evénements* du type référencé sont générées pour les *Clients* abonnés. L'*Invocation de Programme* peut être utilisée comme *EventNotifier* (*Notificateur d'Événements*) de ces *Evénements* ou les notifications peuvent être fournies par un *Objet* principal ou l'*Objet* de *Serveur.*

Les *ProgramTransitionEventTypes* permettent de fournir les données de résultat et de confirmer les transitions d'états aux *Clients* abonnés sur chaque *Transition d'états du Programme* définie. L'*AuditProgramTransitionEventType* permet d'auditer les modifications apportées à l'état de *Programme* conjointement aux *Appels de Méthode* du *Client.*

5.2.5.2 ProgramTransitionEventType

Le *ProgramTransitionEventType* (*Type d'Événement de Transition d'un Programme*) est un sous-type du *TransitionEventType*. Il est utilisé avec les Programmes pour acquérir les résultats intermédiaires ou finaux ou d'autres données associées à une transition d'états. Un Programme peut avoir une définition de *ProgramTransitionEventType* unique pour toute transition. Chaque *ProgramTransitionEventType* spécifie les données d'IntermediateResult (Résultat Intermédiaire) spécifiques à la transition d'états désignée sur le *Type de Programme* considéré. Chaque transition peut produire différentes données de résultats intermédiaires. Le Tableau 9 spécifie le *ProgramTransitionEventType*.

Le Tableau 10 identifie les *ProgramTransitionEventTypes* spécifiés pour les *ProgramTypes* (*Types de Programmes*).

Tableau 9 – ProgramTransitionEventType

Attribut	Valeur				
BrowseName	ProgramTransitionEventType				
IsAbstract	Vrai				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
Sous-type du *TransitionEventType* de base défini dans l'IEC 62541-5:2015, B.4.16.					
HasComponent	Objet	IntermediateResult		BaseObjectType	Facultative

TransitionNumber identifie la transition d'un *Programme* qui a déclenché *l'Événement*.

FromStateNumber identifie l'état avant la transition d'un *Programme*.

Copyright International Electrotechnical Commission

ToStateNumber identifie l'état après la transition d'un *Programme*.

IntermediateResult est un *Objet* qui regroupe un jeu de *Variables* dont les valeurs sont significatives pour le *Programme* au moment de la transition associée. L'*ObjectType (Type d'Objet)* de *IntermediateResult (Résultat Intermédiaire)* spécifie l'ensemble des *Variables* utilisant un jeu de *Références HasComponent*.

Tableau 10 – ProgramTransitionEvents

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
Effets					
HaltedToReadyEvent					
	HasProperty	TransitionNumber	1	PropertyType	
	HasProperty	FromStateNumber	1	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
ReadyToRunningEvent					
	HasProperty	TransitionNumber	2	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
RunningToHaltedEvent					
	HasProperty	TransitionNumber	3	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
RunningToReadyEvent					
	HasProperty	TransitionNumber	4	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
RunningToSuspended Event					
	HasProperty	TransitionNumber	5	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	4	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
SuspendedToRunning Event					
	HasProperty	TransitionNumber	6	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	

Copyright International Electrotechnical Commission

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
Effets					
	HasComponent	IntermediateResults		ObjectType	Facultatif
SuspendedToHaltedEvent					
	HasProperty	TransitionNumber	7	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
SuspendedToReadyEvent					
	HasProperty	TransitionNumber	8	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif
ReadyToHaltedEvent					
	HasProperty	TransitionNumber	9	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Facultatif

5.2.6 AuditProgramTransitionEventType

AuditProgramTransitionEventType (*Type d'Événement d'Audit de Transition d'un Programme*) est un sous-type du *AuditUpdateStateEventType*. Il est utilisé avec les *Programmes* pour assurer l'audit des transitions d'*États du Programme* associées à toute *Méthode de Commande du Programme* invoquée par le *Client*. *Les Serveurs* doivent générer les *AuditProgramTransitionEvents* s'ils prennent en charge le profil "Audit Server Facet" défini dans l'IEC 62541-7.

Le Tableau 11 spécifie la définition d'*AuditProgramTransitionEventType.*

Tableau 11 – AuditProgramTransitionEventType

Attribut	Valeur				
BrowseName	AuditProgramTransitionEventType				
IsAbstract	Vrai				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
Sous-type du *AuditUpdateStateEventType* défini dans l'IEC 62541-5:–, B.4.17.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *AuditUpdateStateEventType* défini dans l'IEC 62541-5:2015, B.4.17, à l'exception de ce qui est noté ci-dessous.

La *Propriété Status (Statut),* spécifiée dans l'IEC 62541-5:2015, 6.4.3, identifie si la transition d'états a pour cause un appel de Méthode de Commande de Programme (*Statut* mis à VRAI) ou non (*Statut* mis à FAUX).

Copyright International Electrotechnical Commission

SourceName (Nom Source) spécifié dans l'IEC 62541-5:2015, 6.4.2, identifie la *Méthode* provoquant la *(*transition d'un *Programme* lorsqu'elle résulte d'une *ProgramControlMethod* invoquée par le *Client*. Le *SourceName* a pour préfixe "Method/" et le nom de la *ProgramControlMethod*, "Method/Start" par exemple.

La *Propriété ClientUserId,* spécifiée dans l'IEC 62541-5:2015, 6.4.3, , identifie l'utilisateur du *Client* ayant émis la *Méthode de Commande de Programme* si elle est associée à cette transition d'états de *Programme*.

La *Propriété ActionTimeStamp,* spécifiée dans l'IEC 62541-5:2015, 6.4.3,, identifie le moment auquel se produit la transition d'états de *Programme* ayant donné lieu à la génération de l'*Evénement*.

La *Propriété TransitionNumber* est une *Variable* qui identifie la transition qui a déclenché l'*Evénement*.

5.2.7 FinalResultData

L'*ObjectType* (*Type d'Objet*) *FinalResultData* (Données finales de résultat) spécifie les *VariableTypes* (*Types de Variables*) qui sont conservées lorsque le *Programme* a accompli sa fonction. L'*ObjectType* comprend un *HasComponent* pour une *VariableType* de chaque *Variable* qui comprend les données finales de résultat.

5.2.8 ProgramDiagnosticDataType

Cette structure contient des éléments qui détaillent l'activité de *ProgramInvocation* qui peuvent aider à diagnostiquer les problèmes liés au *Programme*. Sa composition est définie dans le Tableau 12.

Tableau 12 – Structure de ProgramDiagnosticDataType

Nom	Type	Description
ProgramDiagnosticDataType	structure	
createSessionId	NodeId	Le *CreateSessionId* comporte le *SessionId* de la *Session* au cours de laquelle l'appel à la *Méthode Create* a été émis pour créer l'*Invocation de Programme*.
createClientName	String	*CreateClientName* est le nom du client de la *Session* ayant créé l'*Invocation de Programme.*
invocationCreationTime	UTCTime	*InvocationCreationTime* identifie le moment où l'*Invocation de Programme* a été créée.
lastTransitionTime	UTCTime	*LastTransitionTime* identifie le moment où la dernière transition d'états de *Programme* a eu lieu.
lastMethodCall	String	*LastMethodCall* identifie la dernière *Méthode de Programme* appelée sur l'*Invocation de Programme.*
lastMethodSessionId	NodeId	*LastMethodSessionId* comporte le *SessionId* de la *Session* au cours de laquelle le dernier appel de Méthode de Commande de Programme pour l'*Invocation de Programme* a été émis.
lastMethodInputArguments	Argument	*LastMethodInputArguments* conservent les valeurs des arguments d'entrée sur le dernier appel de *Méthode du Programme.*
lastMethodOutputArguments	Argument	*LastMethodOutputArguments* conservent les valeurs des arguments de sortie sur le dernier appel de *Méthode du Programme.*
lastMethodCallTime	UTCTime	*LastMethodCallTime* identifie le moment du dernier appel de Méthode pour l'*Invocation de Programme.*
lastMethodReturnStatus	StatusResult	*LastMethodReturnStatus* conserve la valeur du return status pour la dernière *Méthode de Commande de Programme* demandée pour cette *Invocation de Programme.*

Copyright International Electrotechnical Commission

Sa représentation dans l'*AddressSpace* est définie dans le Tableau 13.

Tableau 13 – Définition de ProgramDiagnosticDataType

Attributs	Valeur
BrowseName	ProgramDiagnosticDataType

5.2.9 VariableType ProgramDiagnosticType

Ce *VariableType* regroupe des *Variables* simples en utilisant des *DataTypes* simples qui reflètent les éléments de la structure de ProgramDiagnosticDataType. Ses *DataVariables* ont la même sémantique que celle définie en 5.2.8. Le *VariableType* est défini de manière formelle dans le Tableau 14.

Tableau 14 – VariableType ProgramDiagnosticType

Attributs	Valeur			
BrowseName	ProgramDiagnosticType			
IsAbstract	Faux			
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données / Définition de Type**	**Règle de Modélisation**
Sous-type du BaseDataVariableType défini dans l'IEC 62541-5.				
HasComponent	Variable	CreateSessionId	NodeId	Obligatoire
HasComponent	Variable	CreateClientName	String	Obligatoire
HasComponent	Variable	InvocationCreationTime	UTCTime	Obligatoire
HasComponent	Variable	LastTransitionTime	UTCTime	Obligatoire
HasComponent	Variable	LastMethodCall	String	Obligatoire
HasComponent	Variable	LastMethodSessionId	NodeId	Obligatoire
HasComponent	Variable	LastMethodInputArguments	Argument	Obligatoire
HasComponent	Variable	LastMethodOutputArguments	Argument	Obligatoire
HasComponent	Variable	LastMethodCallTime	UTCTime	Obligatoire
HasComponent	Variable	LastMethodReturnStatus	StatusResult	Obligatoire

Copyright International Electrotechnical Commission

Annexe A
(informative)

Exemple de Programme

A.1 Vue d'ensemble

Cet exemple illustre l'utilisation d'un *Programme* pour gérer un téléchargement de domaine dans un système de commande, comme illustré à la Figure A.1. Le téléchargement nécessite de fractionner le transfert des données opérationnelles de commande d'une mémoire secondaire vers la mémoire locale au sein d'un système de commande.

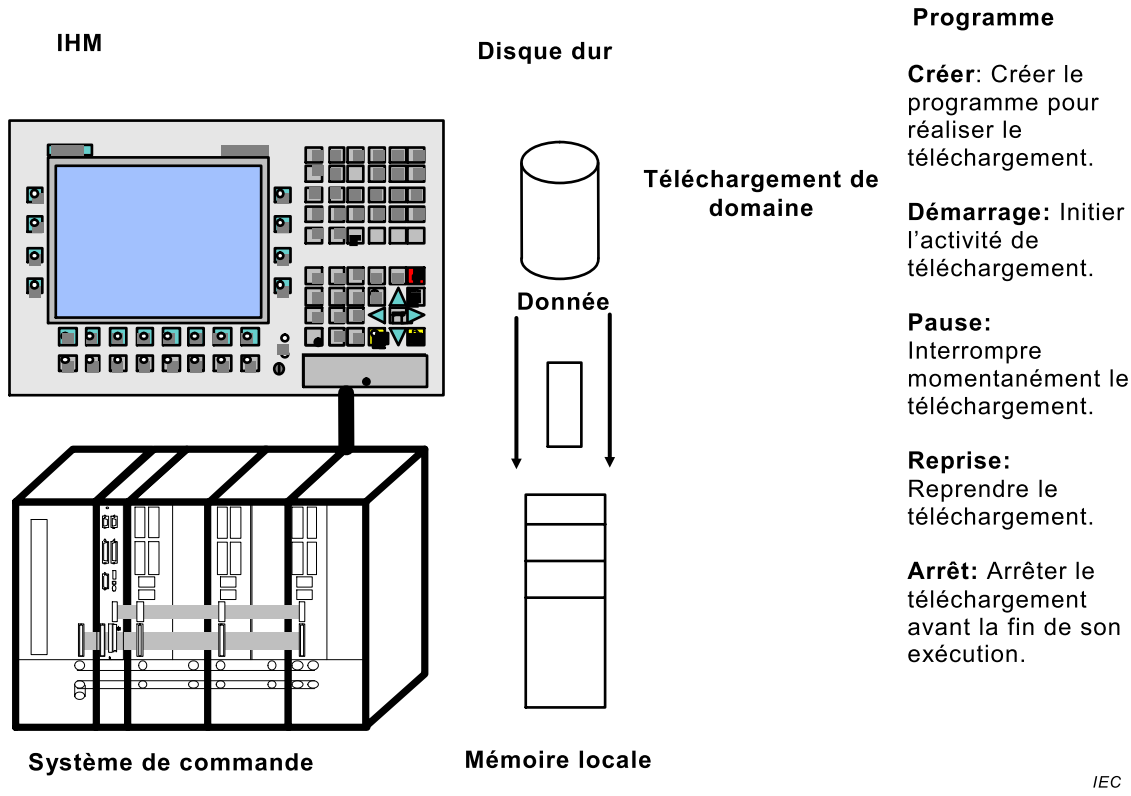

Figure A.1 – Exemple de Programme

Le téléchargement de domaine dispose d'un emplacement source et d'un emplacement cible qui sont identifiés au début du téléchargement. Chaque transfert réussi d'un segment du domaine est indiqué au client qui est également informé de la quantité de données téléchargées. Le client est également avisé de la fin du téléchargement. Le pourcentage des données totales reçues est consigné périodiquement tout au long du téléchargement. En cas d'échec du téléchargement, la cause de la défaillance est consignée. A la fin du téléchargement, les informations relatives à la réalisation de l'opération sont mémorisées sur le *Serveur*.

A.2 Programme DomainDownload

A.2.1 Généralités

Le *Client* utilise le *Programme* "DomainDownload" pour gérer et surveiller le téléchargement d'un domaine sur le *Serveur*.

Copyright International Electrotechnical Commission

A.2.2 Etats de DomainDownload

La Figure A.2 présente le modèle d'état de base du *Programme* DomainDownload. Le *Programme* a trois états primaires, *Ready*, *Running* et *Halted* qui correspondent aux états normalisés d'un *Type de Programme*. De plus, le *DomainDownloadType* étend le *Type de Programme* en définissant des diagrammes d'états finis subordonnés pour les états de *Programme Running* et *Halted*. Les états subordonnés décrivent de manière plus détaillée les opérations de téléchargement et permettent au *Client* de surveiller le déroulement du téléchargement de manière plus précise.

Le *Client* crée une instance (*Invocation de Programme)* d'un *Programme* DomainDownload à chaque réalisation d'un téléchargement. L'instance existe jusqu'à ce qu'elle soit retirée de manière explicite par le *Client*. L'état initial du *Programme* est *Ready* et l'état final est Halted. Le DomainDownload peut être momentanément interrompu, puis repris ou abandonné. Une fois arrêté, le programme ne peut pas être redémarré.

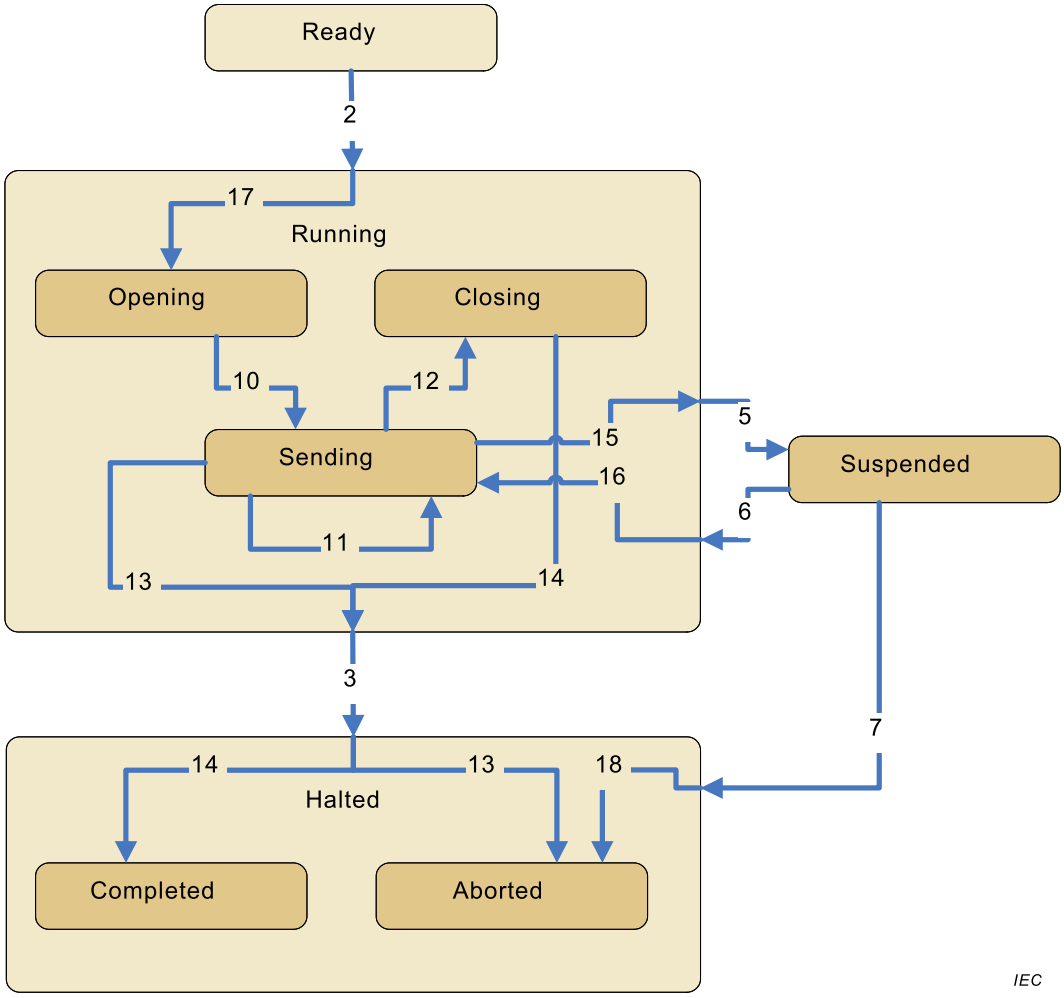

Figure A.2 – Diagramme d'état de DomainDownload

La Figure A.2 illustre la séquence des transitions d'états. Une fois le téléchargement démarré, le *Programme* se déroule jusqu'à l'état *Opening* (*Ouverture*). Après l'ouverture de la source des données, une séquence de transferts se produit à l'état *Sending* (*Transmission*). À la fin du transfert, les *Objets* sont fermés à l'état *Closing* (*Fermeture*). Si le transfert se termine avant la fin du téléchargement de toutes les données ou en cas d'erreur, le téléchargement est arrêté et le *Programme* passe à l'état *Aborted* (*Abandonné*), sinon le *Programme* s'arrête à l'état *Completed* (*Fini*). Le Tableau A.1 présente les états avec les transitions d'états correspondantes.

Copyright International Electrotechnical Commission

A.2.3 Transitions de DomainDownload

Les transitions d'états valides spécifiées pour le *Programme* DomainDownload sont indiquées dans le Tableau A.1. Chaque transition définit un état de début et un état de fin pour la transition et est identifiée par un numéro unique. Cinq transitions sont issues du *Type du Programme* de base et conservent les numéros d'identificateur de transition spécifiés pour les *Programmes*. Les transitions supplémentaires concernent les états du *Programme* de base avec les états subordonnés définis pour le DomainDownload. Des numéros d'identificateur de transition uniques différents des identificateurs de transition d'un *Programme* de base ont été attribués à ces états. Lorsque des transitions ont lieu entre des sous-états et les états du *Programme* de base, deux transitions sont spécifiées. Une transition identifie la modification de l'état de base et l'autre la modification du sous-état. Par exemple, les transitions *ReadytoRunning* et *ToOpening* se produisent en même temps.

Le tableau spécifie également les états définis, les causes des transitions et les effets de chaque transition. Le *Client* utilise les Méthodes de Commande du *Programme* pour "exécuter" le DomainDownload. Les *Méthodes* provoquent ou déclenchent les transitions spécifiées. Les effets des transitions sont les *EventTypes* spécifiés qui informent le *Client* du déroulement du *Programme*.

Tableau A.1 – Etats de DomainDownload

N°	Dénomination de transition	Cause	De l'Etat	À l'Etat	Effet
2	ReadyToRunning	Méthode Start	Ready	Running	Déclenche la Transition 2 Événement/Résultat
3	RunningToHalted	Méthode Halt/Erreur ou Interne	Running	Halted	Déclenche la Transition 3 Événement/Résultat
5	RunningToSuspended	Méthode Suspend	Running	Suspended	Déclenche la Transition 5 Événement/Résultat
6	SuspendedToRunning	Méthode Resume	Suspended	Running	Déclenche la Transition 6 Événement/Résultat
7	SuspendedToHalted	Méthode Halt	Suspended	Halted	Déclenche la Transition 7 Événement/Résultat
10	OpeningToSending	Interne	Opening	Sending	Déclenche la Transition 10 Événement/Résultat
11	SendingToSending	Interne	Sending	Sending	Déclenche la Transition 11 Événement/Résultat
12	SendingToClosing	Interne	Sending	Closing	Déclenche la Transition 12 Événement/Résultat
13	SendingToAborted	Méthode Halt/Erreur	Opening	Aborted	Déclenche la Transition 13 Événement/Résultat
14	ClosingToCompleted	Interne	Closing	Completed	Déclenche la Transition 14 Événement/Résultat
15	SendingToSuspended	Méthode Suspend	Sending	Suspended	Déclenche la Transition 16 Événement/Résultat
16	SuspendedToSending	Méthode Resume	Suspended	Sending	Déclenche la Transition 17 Événement/Résultat
18	SuspendedToAborted	Méthode Halt	Suspended	Aborted	Déclenche la Transition 18 Événement/Résultat
17	ToOpening	Interne	Ready	Opening	Déclenche la Transition 19 Événement/Résultat

Copyright International Electrotechnical Commission

A.2.4 Méthodes de DomainDownload

A.2.4.1 Généralités

Quatre *Méthodes* du Programme normalisées sont spécifiées pour l'exécution du *Programme* DomainDownload: *Start, Suspend, Resume* et *Halt.* Aucune *Méthode* supplémentaire n'est spécifiée. Les comportements de base de ces *Méthodes* sont définis par le *Type du Programme.* La *Méthode Start* initie l'opération de téléchargement et transmet les emplacements source et destination pour transfert. La *Méthode Suspend* permet d'interrompre momentanément l'activité. La *Méthode Resume* relance le téléchargement, après l'interruption. La *Méthode Halt* permet d'abandonner le téléchargement. Chacune des *Méthodes* provoque une transition d'états du *Programme* et une transition de sous-états. La transition d'états particulière dépend de l'état courant au moment de l'appel de la *Méthode*. Si un *Appel de Méthode* est réalisé lorsque le DomainDownload est à un état dans lequel la *Méthode* considérée n'a pas de transition associée, la *Méthode* renvoie un statut d'erreur indiquant l'état non valide pour la *Méthode*.

A.2.4.2 Arguments de Méthode

Lors de son appel, la *Méthode Start* spécifie trois arguments d'entrée à transmettre: Domain Name (Nom de Domaine), DomainSource (Source de Domaine) et DomainDestination (Destination de Domaine). Les autres *Méthodes* ne nécessitent aucun argument d'entrée. Aucun argument de sortie n'est spécifié pour les *Méthodes* de DomainDownload. Le statut d'erreur résultant pour le *Programme* fait partie du *Service Appel.*

A.2.5 Événements de DomainDownload

A.2.5.1 Généralités

Un *ProgramTransitionEventType* est spécifié pour chacune des transitions du *Programme* DomainDownload. Les *Types d'Evénements* déclenchent une notification d'*Evénement* particulière transmise au *Client* lorsque la transition d'états associée se produit en cours d'exécution de l'Instance d'un *Programme*. La notification d'*Evénement* identifie la transition. La transition d'états SendingToSending comprend également les données de résultat intermédiaires.

A.2.5.2 Informations concernant les Evénements

L'*Evénement* de transition d'un *Programme* SendingToSending relaie les données de résultat intermédiaires vers le *Client* avec la notification. A chaque transition, les éléments de données décrivant la quantité et le pourcentage de données transférées sont transmis au *Client*.

A.2.5.3 Données finales de Résultat

Le *Programme* DomainDownload conserve les données finales de résultat après un téléchargement terminé ou abandonné. Les données comprennent le temps de transaction total et la taille du domaine. Dans le cas d'un téléchargement abandonné, la raison de cet arrêt est conservée.

A.2.6 Modèle de DomainDownload

A.2.6.1 Vue d'ensemble

Le modèle OPC UA du *Programme* DomainDownload est présenté dans les tableaux et figures ci-après qui dans leur ensemble définissent les composants constitutifs de ce *Programme*. Pour des raisons de clarté, les figures présentent une progression de parties du modèle qui complètent le contenu des tableaux et illustrent la composition du *Programme.*

Copyright International Electrotechnical Commission

La définition de type du *Programme* DomainDownload représente avec précision le comportement du *Programme* en termes de composants OPC UA. Ces composants peuvent être explorés par un *Client* afin d'interpréter ou de valider les actions du *Programme*.

A.2.6.2 DomainDownloadType

Le DomainDownloadType (Type de DomainDownload) est un sous-type déduit du *Type de Programme*. Il spécifie l'utilisation ou non des composants facultatifs de *Type de Programme*, des extensions valides telles que les diagrammes d'états finis subordonnés, ainsi que des valeurs d'attribut contraintes appliquées aux instances des *Programmes* DomainDownload.

Le Tableau A.2 spécifie les composants facultatifs et étendus définis par le Type de DomainDownload. A noter les références aux deux Types de diagrammes de sous-états finis, TransferStateMachine et FinishStateMachine. Le DomainDownloadType omet les références à la Méthode de Commande du Programme Reset et sa transition d'états associée (HaltedToReady), qu'il ne prend pas en charge.

Tableau A.2 – Type de DomainDownload

Attribut	**Valeur**				
	Comprend tous les attributs non facultatifs spécifiés pour le Type du Programme				
BrowseName	DomainDownloadType				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasComponent	Objet	TransferStateMachine		StateMachineType	Obligatoire
HasComponent	Objet	FinishStateMachine		StateMachineType	Obligatoire
HasComponent	Variable	ProgramDiagnostic		ProgramDiagnosticType	Obligatoire
HasComponent	Objet	ReadyToRunning		TransitionType	Obligatoire
HasComponent	Objet	RunningToHalted		TransitionType	Obligatoire
HasComponent	Objet	RunningToSuspended		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToRunning		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToHalted		TransitionType	Obligatoire
HasComponent	Méthode	Start			Obligatoire
HasComponent	Méthode	Suspend			Obligatoire
HasComponent	Méthode	Halt			Obligatoire
HasComponent	Méthode	Resume			Obligatoire
HasComponent	Objet	FinalResultData		BaseObjectType	Obligatoire

Le Tableau A.3 spécifie le Type de Diagramme d'États Finis Transfer (Transfert) qui est un diagramme de sous-états finis du *Type du Programme* DomainDownload. Cette définition identifie les *Types d'États* qui constituent les sous-états du *Type d'État* du *Programme* Running (Exécution).

Copyright International Electrotechnical Commission

Tableau A.3 – Type de Diagramme d'Etats Finis Transfer

Attribut	Valeur				
	Comprend tous les attributs spécifiés pour le FiniteStateMachineType				
BrowseName	TransferStateMachineType				
IsAbstract	Faux				
Références	**Classe de nœud**	**Nom de navigation**	**Type de données**	**Définition de type**	**Règle de modélisation**
HasComponent	Objet	Opening		StateType	Obligatoire
HasComponent	Objet	Sending		StateType	Obligatoire
HasComponent	Objet	Closing		StateType	Obligatoire
HasComponent	Objet	ReadyToOpening		TransitionType	Obligatoire
HasComponent	Objet	OpeningToSending		TransitionType	Obligatoire
HasComponent	Objet	SendingToClosing		TransitionType	Obligatoire
HasComponent	Objet	SendingToAborted		TransitionType	Obligatoire
HasComponent	Objet	SendingToSuspended		TransitionType	Obligatoire
HasComponent	Objet	SuspendedToSending		TransitionType	Obligatoire
HasComponent	Méthode	Start			Obligatoire
HasComponent	Méthode	Suspend			Obligatoire
HasComponent	Méthode	Halt			Obligatoire
HasComponent	Méthode	Resume			Obligatoire

Le Tableau A.3 spécifie les *StateTypes* (*Types d'États*) associés au Type de Diagramme d'États Finis Transfer. Tous ces états sont des sous-états de l'état *Running* du *Type de Programme* de base.

L'état Opening représente l'état de préparation pour le téléchargement de domaine.

L'état Sending représente l'état d'activité pour le transfert des données de la source à la destination.

L'état Closing représente la phase de "nettoyage" du téléchargement.

Le Tableau A.4 définit les états du *TransferStateMachineType*.

Copyright International Electrotechnical Commission

Tableau A.4 – Diagramme d'Etats Finis Transfer – États

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
États					
Opening	HasProperty	StateNumber	5	PropertyType	
	ToTransition	OpeningToSending		TransitionType	
	FromTransition	ToOpening		TransitionType	
	ToTransition	OpeningToSending		TransitionType	
Sending	HasProperty	StateNumber	6	PropertyType	
	FromTransition	OpeningToSending		TransitionType	
	ToTransition	SendingToSending		TransitionType	
	ToTransition	SendingToClosing		TransitionType	
	ToTransition	SendingToSuspended		TransitionType	
	FromTransition	ToSending		TransitionType	
Closing	HasProperty	StateNumber	7	PropertyType	
	ToTransition	ClosingToCompleted		TransitionType	
	ToTransition	ClosingToAborted		TransitionType	
	FromTransition	SendingToClosing		TransitionType	

Le Tableau A.5 spécifie le *Type de Diagramme d'Etats Finis* qui est un diagramme de sous-états finis du *Type du Programme* DomainDownload. Cette définition identifie les *Types d'États* qui constituent les sous-états du *Type d'État* du *Programme* Halted.

Tableau A.5 – Type de Diagramme d'Etats Finish

Attribut	Valeur				
	Comprend tous les attributs spécifiés pour le FiniteStateMachineType				
BrowseName	TransferStateMachineType				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasComponent	Objet	Completed		StateType	Obligatoire
HasComponent	Objet	Aborted		StateType	Obligatoire

Le Tableau A.6 spécifie les *Types d'États* associés au *Type de Diagramme d'États Finis*. À noter qu'il s'agit des états finaux et qu'ils n'ont pas de transitions associées entre eux.

Copyright International Electrotechnical Commission

Tableau A.6 – Diagramme d'Etats Finish – États

Nom de navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Cible	NOTES
États					
Aborted	HasProperty	StateNumber	8	PropertyType	
	FromTransition	OpeningToAborted		TransitionType	
	FromTransition	ClosingToAborted		TransitionType	
Completed	HasProperty	StateNumber	9	PropertyType	
	FromTransition	ClosingToCompleted		TransitionType	

L'état *Aborted* représente l'état final qui indique une opération de téléchargement de domaine incomplète ou ayant échoué.

L'état *Completed* représente l'état final qui indique une opération de téléchargement de domaine réussie.

Le Tableau A.7 spécifie le comportement de contrainte d'un DomainDownload.

Tableau A.7 – Valeurs des variables de propriétés et d'attributs du type DomainDownload

Classe de Nœud	Nom de Navigation	Type de Données	Valeur de Données	Règle de Modélisation
Variable	Creatable	Boolean	Vrai	--
Variable	Deletable	Boolean	Vrai	Obligatoire
Variable	AutoDelete	Boolean	Faux	Obligatoire
Variable	RecycleCount	Int32	0	Obligatoire
Variable	InstanceCount	UInt32	PropertyType	--
Variable	MaxInstanceCount	UInt32	500	--
Variable	MaxRecycleCount	UInt32	0	--

Une *Invocation de Programme* DomainDownload peut être créée mais également supprimée par un *Client*. L'*Invocation de Programme* ne se supprime pas elle-même lorsqu'elle est arrêtée et demeure jusqu'à ce que le *Client* la retire de manière explicite. Une *Invocation de Programme* DomainDownload ne peut pas être réinitialisée à des fins de redémarrage. Le *Serveur* prend en charge simultanément jusqu'à 500 *Invocations de Programme* DomainDownload.

La Figure A.3 présente un modèle partiel de DomainDownloadType qui illustre l'association entre les états et les Diagrammes d'États Finish, Transfer et DomainDownload. À noter que le numéro d'état courant des diagrammes de sous-états finis n'est valide que lorsque l'état de base DomainDownload actif fait référence au diagramme de sous-états finis, Running pour l'état courant Transfer (Transfert) et Halted pour l'état courant Finish (Fini).

Copyright International Electrotechnical Commission

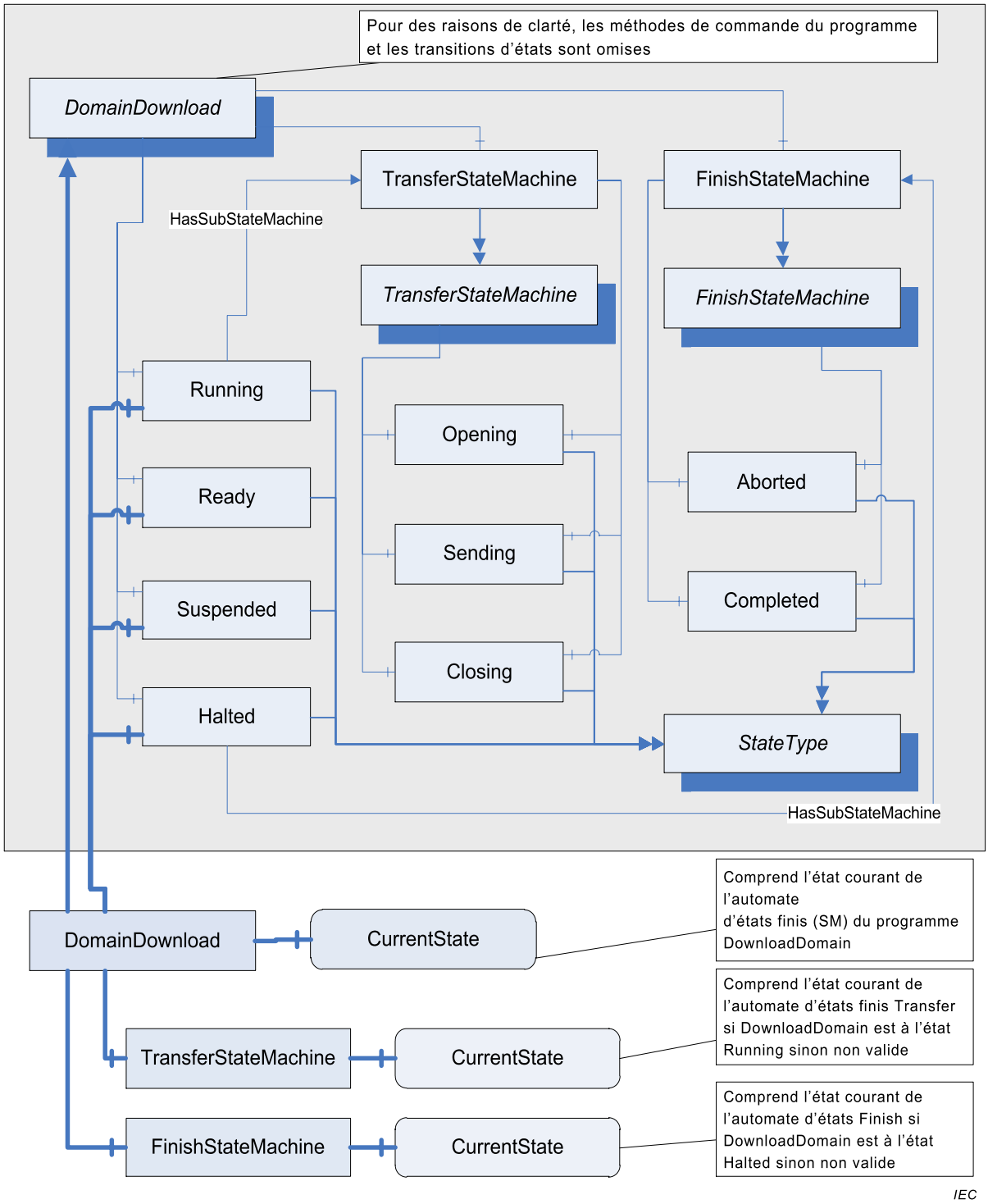

Figure A.3 – Modèle d'état partiel de DomainDownloadType

Le Tableau A.8 spécifie les *ProgramTransitionTypes* définis en complément aux *ProgramTransitionTypes* spécifiés pour les *Programmes* dans le Tableau 7. Ces types associent les états des diagrammes de sous-états finis Transfer et Finish aux états du *Programme* de base.

Copyright International Electrotechnical Commission

Tableau A.8 – Types de transitions supplémentaires de DomainDownload

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Dible	NOTES
Transitions					
ToSending	HasProperty	TransitionNumber	10	PropertyType	
	ToState	Sending		StateType	
	FromState	Opening		StateType	
	HasCause	Start			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SendingToSending	HasProperty	TransitionNumber	11	PropertyType	
	ToState	Sending		StateType	
	FromState	Sending		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToClosing	HasProperty	TransitionNumber	12	PropertyType	
	ToState	Closing		StateType	
	FromState	Sending		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToAborted	HasProperty	TransitionNumber	13	PropertyType	
	ToState	Aborted		StateType	
	FromState	Sending		StateType	
	HasCause	Halt			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ClosingToCompleted	HasProperty	TransitionNumber	14	PropertyType	
	ToState	Completed		StateType	
	FromState	Closing		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToSuspended	HasProperty	TransitionNumber	15	PropertyType	
	ToState	Suspended		StateType	
	FromState	Sending		StateType	
	HasCause	Suspend			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToSending	HasProperty	TransitionNumber	16	PropertyType	
	ToState	Sending		StateType	

Copyright International Electrotechnical Commission

Nom de Navigation	Références	Nom de Navigation Cible	Valeur	Définition de Type Dible	NOTES
Transitions					
	FromState	Suspended		StateType	
	HasCause	Resume			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedTo Aborted	HasProperty	TransitionNumber	18	PropertyType	
	ToState	Aborted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
ReadyToOpening	HasProperty	TransitionNumber	17	PropertyType	
	ToState	Opening		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Méthode
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			

Les Figure A.4 à Figure A.10 illustrent des parties du modèle de DomainDownloadType. Dans chaque figure, les états, *Méthodes*, transitions et *EventTypes* référencés sont identifiés pour une ou deux transitions d'états.

Copyright International Electrotechnical Commission

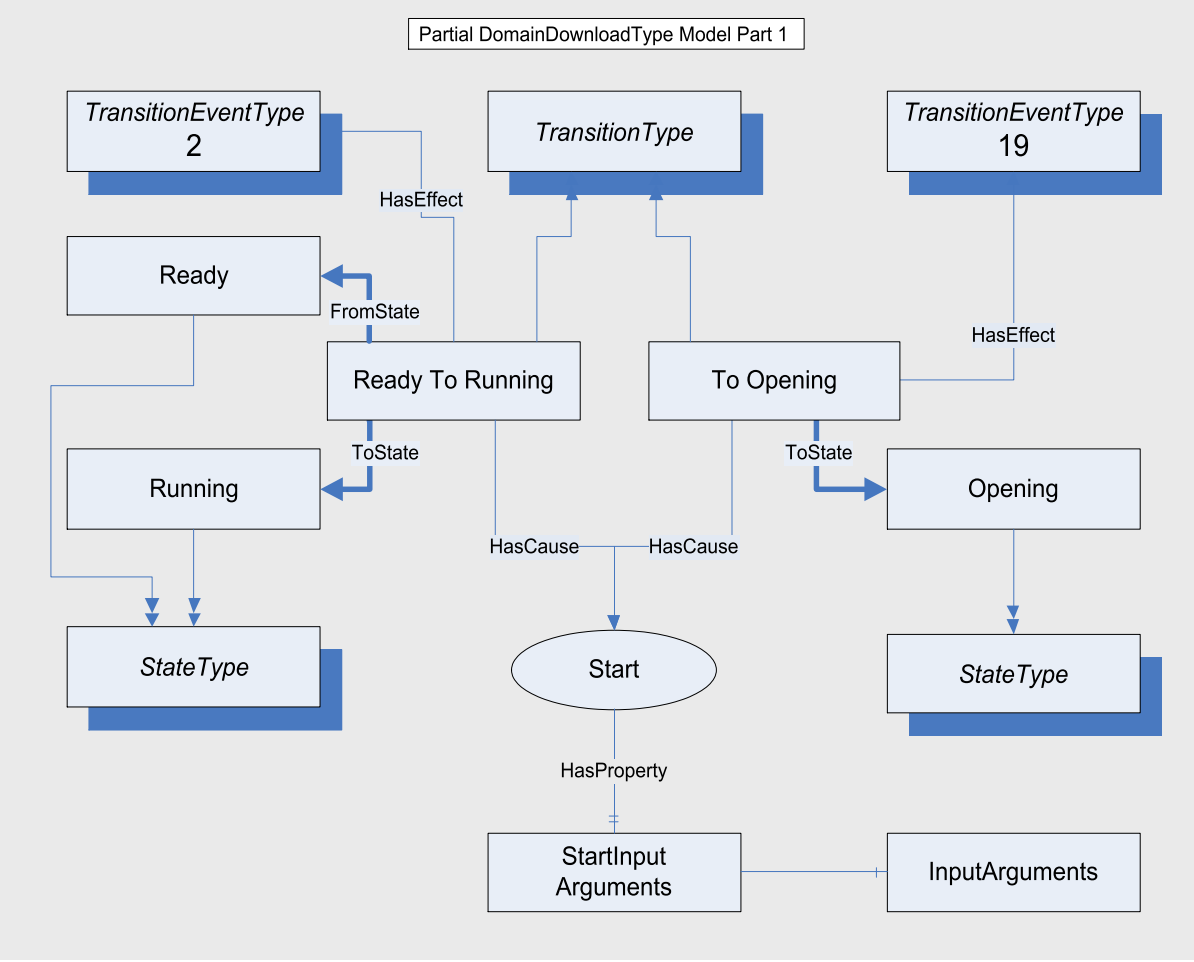

Figure A.4 – Modèle ReadyToRunning

La Figure A.4 illustre le modèle applicable à la transition d'un *Programme* ReadyToRunning. La transition est provoquée par la *Méthode Start*. La *Méthode Start* nécessite trois arguments d'entrée. Le *Client* utilise le service *Appel de Méthode* pour invoquer la *Méthode Start* et transmettre les arguments. Lorsqu'elle aboutit, l'*Invocation de Programme* passe à l'état Running et à l'état subordonné Transfer Opening (Ouverture de Transfert). Le *Serveur* émet deux notifications d'*Evénements*, ReadyToRunning (2) et ToOpening (19).

Tableau A.9 – Ajouts de la Méthode Start

Attributs	**Valeur**				
BrowseName	Start				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasProperty	Variable	InputArgument	Argument[]	PropertyType	–

Le Tableau A.9 spécifie que la *Méthode Start* pour le *DomainDownloadType* nécessite des arguments d'entrée. Le Tableau A.10 identifie les *Arguments Start* nécessaires.

Copyright International Electrotechnical Commission

Tableau A.10 – StartArguments

Dénomination	Type	Valeur
Argument 1	structure	
name	String	SourcePath
dataType	NodeId	StringNodeId
valueRank	Int32	-1 (-1 = scalar)
arrayDimensions	UInt32[]	null
description	LocalizedText	Le spécificateur de source pour le domaine
Argument 2	structure	
Name	String	DesinationPath
dataType	NodeId	StringNodeId
valueRank	Int32	-1 (-1 = scalar)
arrayDimensions	UInt32[]	null
description	LocalizedText	Le spécificateur de destination pour le domaine
Argument 3	structure	
name	String	DomainName
dataType	NodeId	StringNodeId
arrayDimensions	UInt32[]	null
valueRank	Int32	-1 (-1 = scalar)
description	LocalizedText	Le nom du domaine

La Figure A.5 illustre le modèle applicable aux transitions du *Programme* OpeningToSending et SendingToClosing. Comme l'indique le tableau des transitions, ces transitions d'états ne nécessitent aucune application de *Méthodes*, car elles sont pilotées par les actions internes du *Serveur*. Des notifications d'*Evénements* sont générées à chaque transition d'états (10 à 12) lorsqu'elles se produisent.

Copyright International Electrotechnical Commission

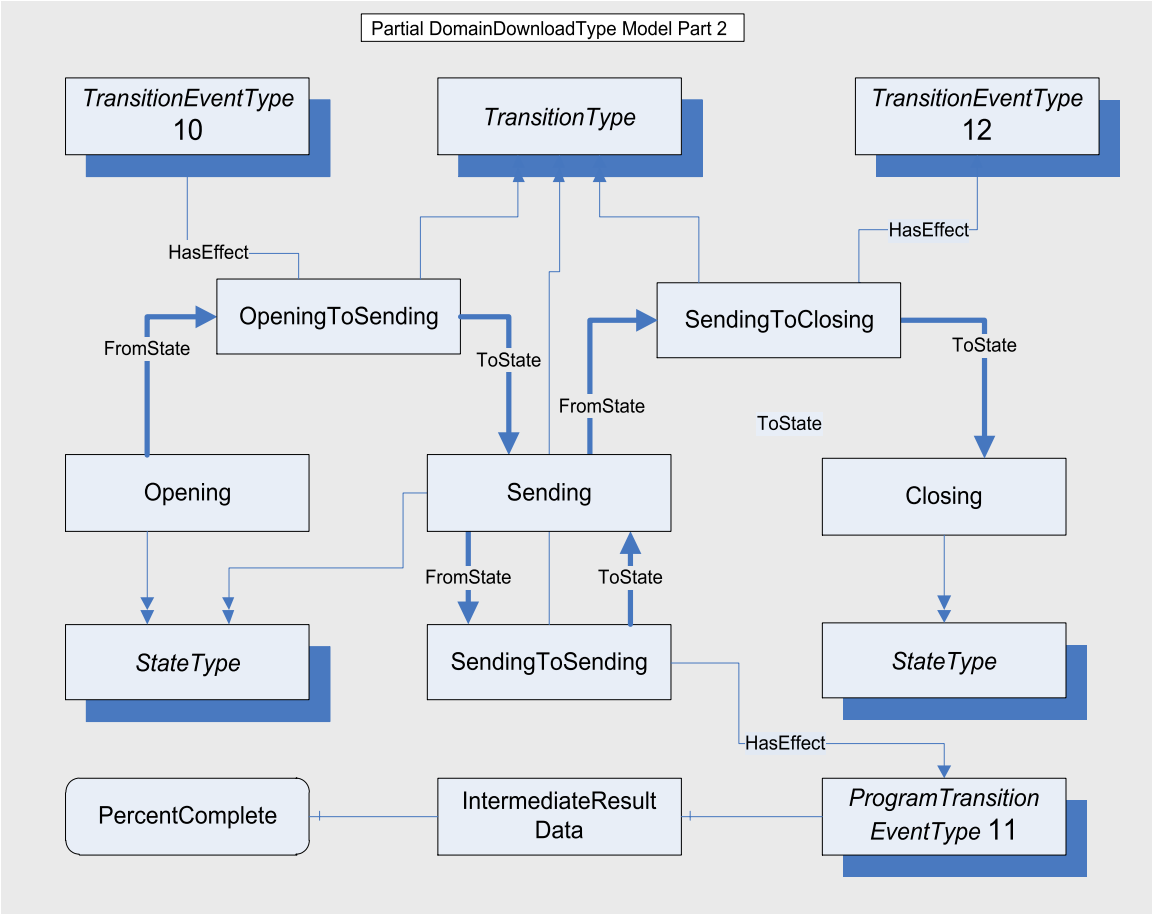

Figure A.5 – Modèle OpeningToSending ToClosing

A noter qu'une transition d'états peut démarrer et se terminer au même état (Sending). Dans ce cas, la transition a un objet particulier. L'effet du *ProgramTransitionEventType* référencé par la transition d'états SendingToSending a une *Référence d'Objet* I*ntermediateResultData*. L'*Objet IntermediateResultData* permet d'identifier deux *Variables* dont les valeurs sont obtenues à chaque occurrence d'une transition d'états. Les valeurs sont transmises au *Client* avec la notification d'*Evénement*. Le Tableau A.11 définit l'*ObjectType IntermediateResults* et le Tableau A.12 définit les *Variables* d'*ObjectType*.

Tableau A.11 – Objet Résultats intermédiaires

Attribut	**Valeur**				
	Comprend tous les attributs spécifiés pour l'ObjectType				
BrowseName	IntermediateResults				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasComponent	Variable	AmountTransferred	Long	VariableType	Obligatoire
HasComponent	Variable	PercentageTransferred	Long	VariableType	Obligatoire

Copyright International Electrotechnical Commission

Tableau A.12 – Variables des données de résultat intermédiaires

Variables de Résultat Intermédiaire	Type	Valeur
Variable 1	Structure	
Name	String	AmountTransferred
dataType	NodeId	StringNodeId
description	LocalizedText	Octets des données de domaine transférées.
Variable 2	Structure	
Name	String	PercentageTransferred
dataType	NodeId	StringNodeId
description	LocalizedText	Pourcentage des données de domaine transférées.

La Figure A.6 illustre le modèle applicable à la transition d'états RunningToSuspended. La cause de cette transition est la *Méthode Suspend.* Le *Client* peut interrompre le téléchargement des données de domaine à la commande. La transition de Running à Suspended entraîne la génération d'*Evénements* de *TransitionEventTypes* 5 et 16. A noter qu'il n'existe plus d'état courant valide pour le Diagramme d'États Finis Transfer.

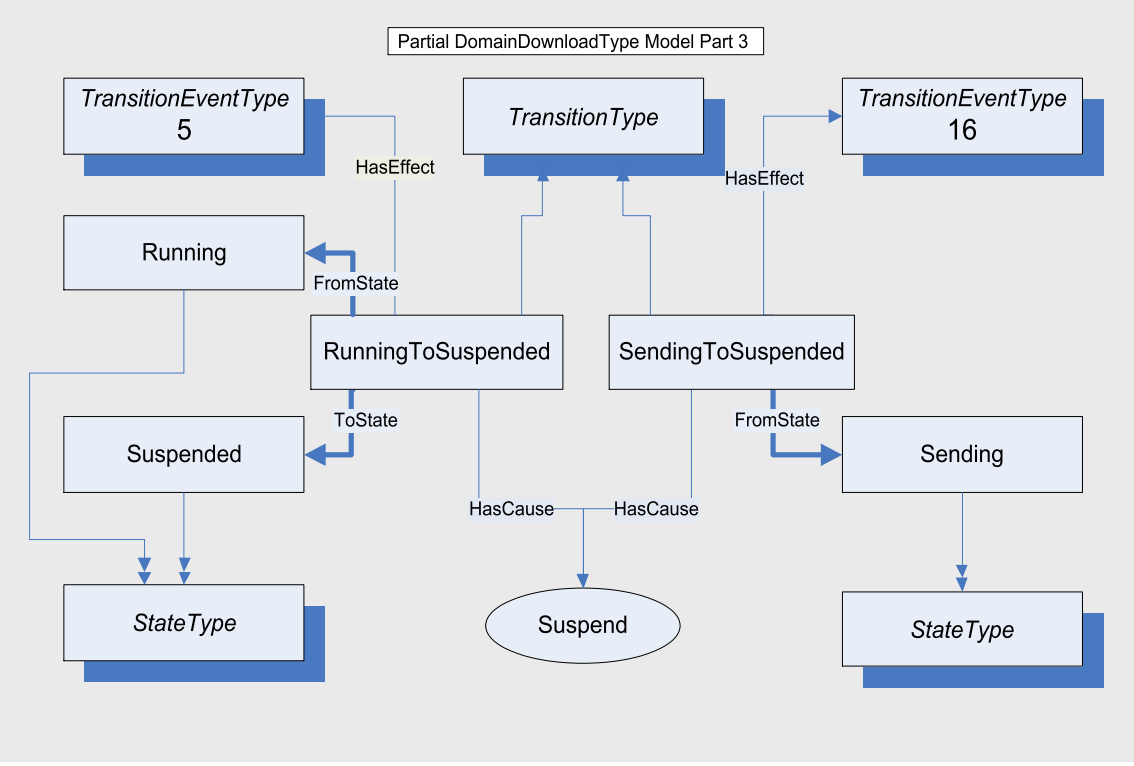

Figure A.6 – Modèle RunningToSuspended

La Figure A.7 illustre le modèle applicable à la transition d'états SuspendedToRunning. La cause de cette transition est la *Méthode Resume.* Le *Client* peut reprendre le téléchargement des données de domaine à la commande. La transition de Suspended à Running génère l'*Evénement* de *TransitionEventTypes* 6 et 17. Désormais l'état Running est actif, et l'état Sending du Diagramme d'États Finis Transfer est de nouveau spécifié pour le *CurrentStateNumber*.

Copyright International Electrotechnical Commission

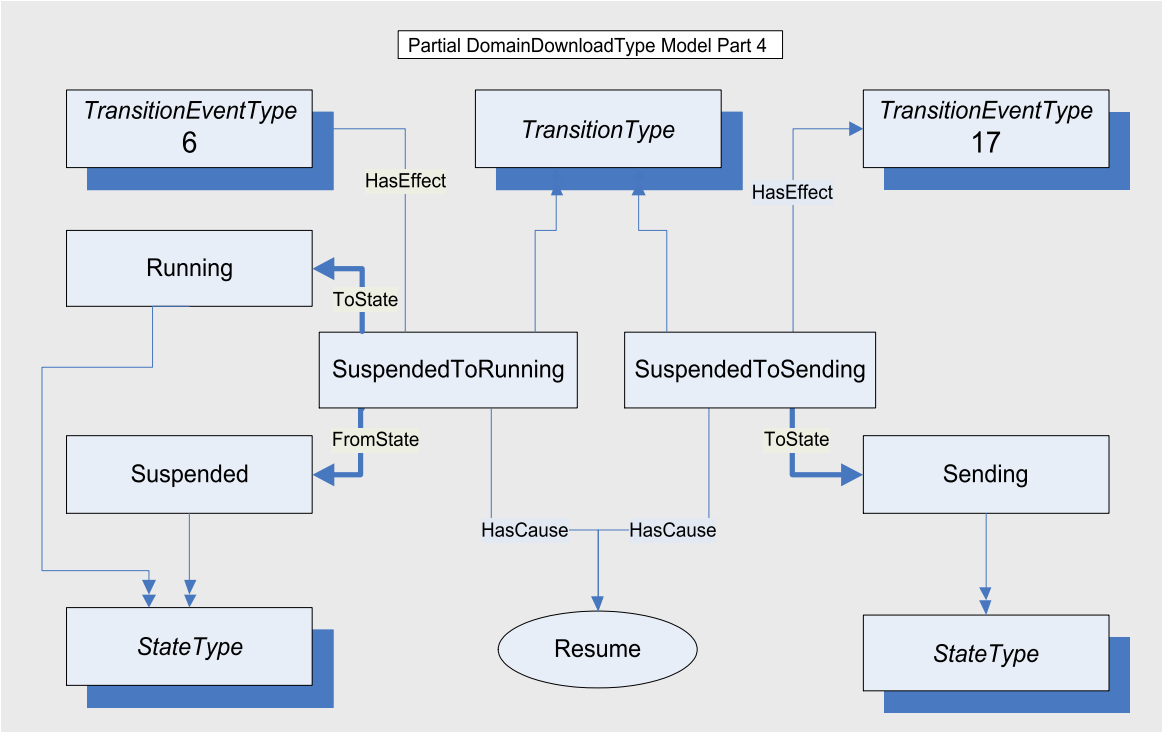

Figure A.7 – Modèle SuspendedToRunning

La Figure A.8 illustre le modèle applicable à la transition d'états RunningToHalted en cas d'achèvement anormal du téléchargement de domaine. La cause de cette transition est la *Méthode Halt*. Le *Client* peut terminer le téléchargement des données de domaine à la commande. La transition de Running à Halted génère l'*Evénement* de *TransitionEventTypes* 3 et 15. Le *TransitionEventType* 15 indique la transition depuis l'état Sending à la fin de l'État Running et la transition à l'état Aborted au passage à l'état Halted.

Copyright International Electrotechnical Commission

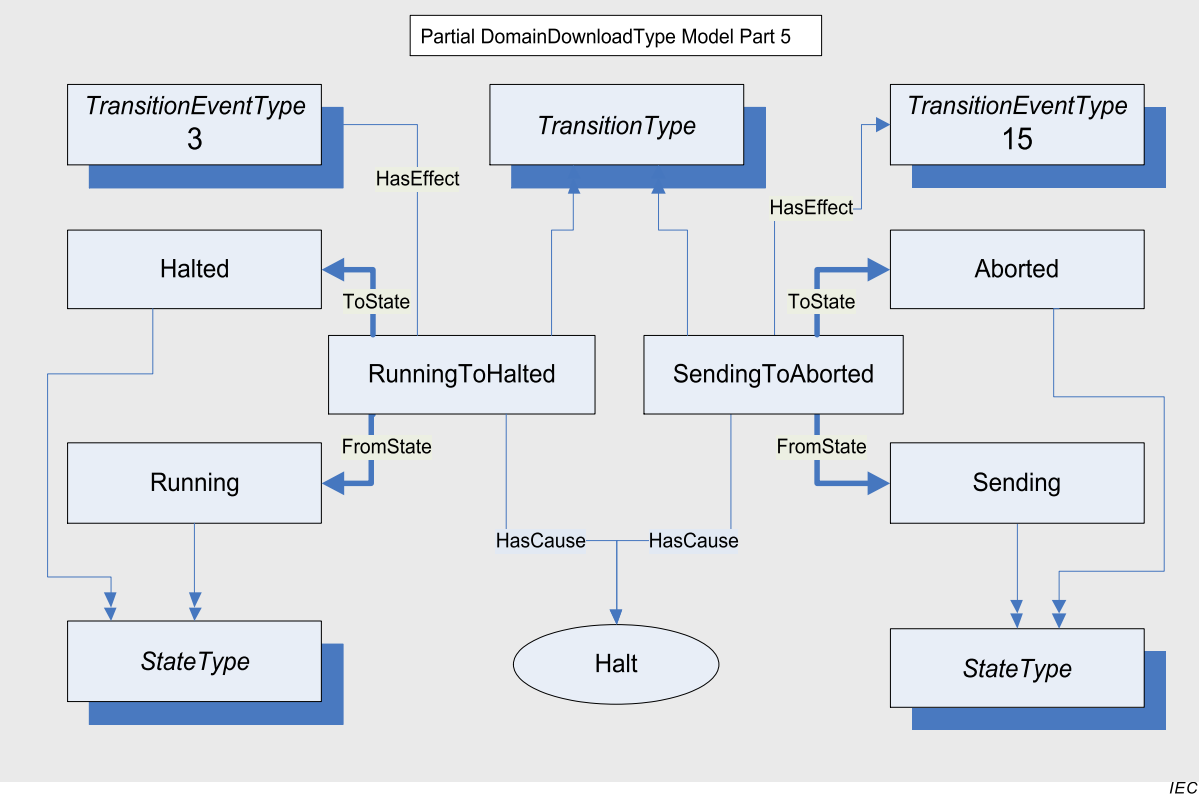

Figure A.8 – Modèle Running To Halted – Aborted

La Figure A.9 illustre le modèle applicable à la transition d'états SuspendedToHalted en cas d'achèvement anormal du téléchargement de domaine. La cause de cette transition est la *Méthode Halt.* Le *Client* peut terminer le téléchargement des données de domaine à la commande lorsqu'il est interrompu. La transition de Suspended à Halted génère des notifications d'*Evénements* de *TransitionEventTypes* 7 et 18.

Copyright International Electrotechnical Commission

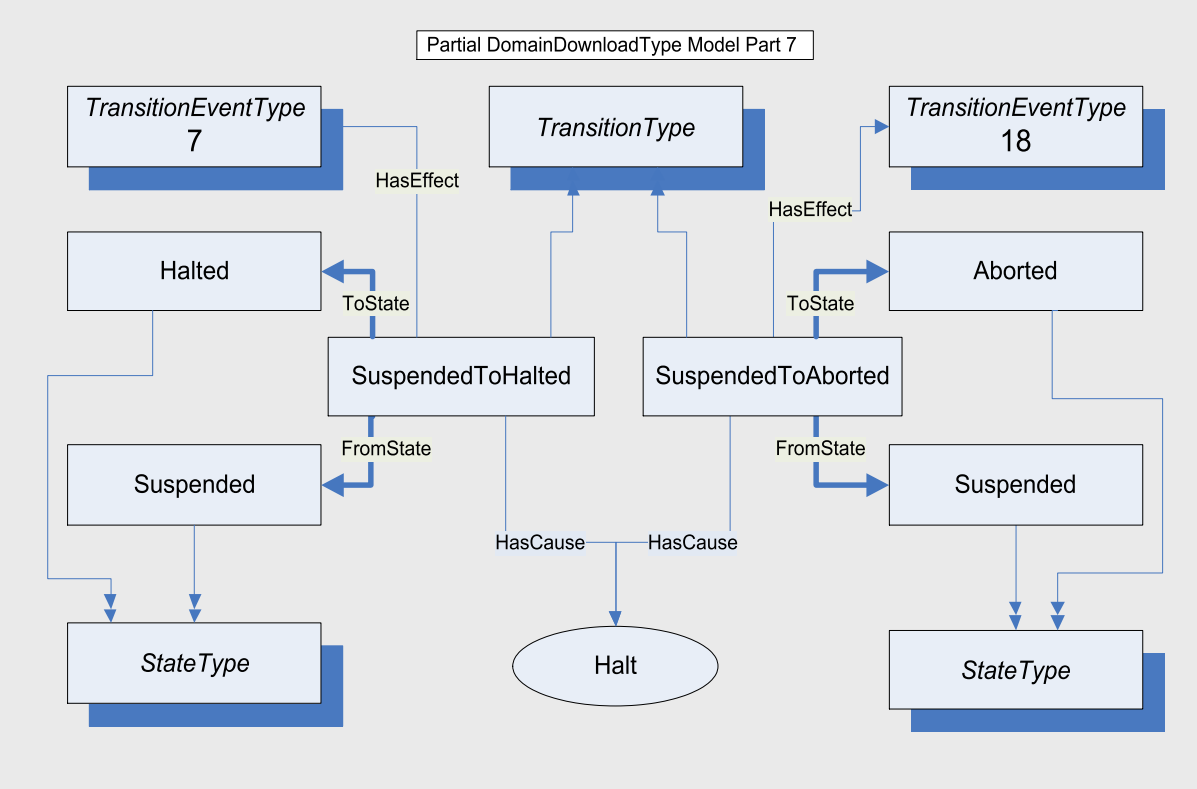

Figure A.9 – Modèle Suspended To Aborted

La Figure A.10 illustre le modèle applicable à la transition d'états RunningToCompleted en cas d'achèvement normal du téléchargement de domaine. La cause de cette transition est interne. La transition de Closing à Halted génère l'Evénement de *TransitionEventTypes* 3 et 14. Le *TransitionEventType* 14 indique la transition depuis l'état Closing à la fin de l'état Running et la transition à l'état Completed au passage à l'état Halted.

Le DomainDownloadType comprend une référence de composant à un *Objet FInalResultData*. Cet *Objet* fait référence aux *Variables* qui conservent les informations relatives au téléchargement de domaine lorsqu'il est terminé. Ces données peuvent être lues par les *Clients* qui ne sont pas abonnés aux notifications d'*Evénements*. Les données de résultat sont décrites dans le Tableau A.13 et les *Variables* dans le Tableau A.14.

Tableau A.13 – Données Finales de Résultat

Attribut	**Valeur**				
	Comprend tous les attributs spécifiés pour l'ObjectType				
BrowseName	FinalResultData				
IsAbstract	Faux				
Références	**Classe de Nœud**	**Nom de Navigation**	**Type de Données**	**Définition de Type**	**Règle de Modélisation**
HasComponent	Variable	DownloadPerformance	Long	VariableType	Obligatoire
HasComponent	Variable	FailureDetails	Long	VariableType	Obligatoire

Le débit net de transfert des données de téléchargement de domaine et la raison détaillée justifiant l'abandon de téléchargements sont consignés comme les données finales de résultat pour chaque *Invocation de Programme.*

Copyright International Electrotechnical Commission

Tableau A.14 – Variables finales de résultat

Variables Finales de Résultat	Type	Valeur
Variable 1	Structure	
Name	String	DownloadPerformance
dataType	NodeId	Double
description	LocalizedText	Débit des données de domaine transférées
Variable 2	Structure	
Name	String	FailureDetails
dataType	NodeId	StringNodeId
description	LocalizedText	Description de la raison justifiant l'abandon

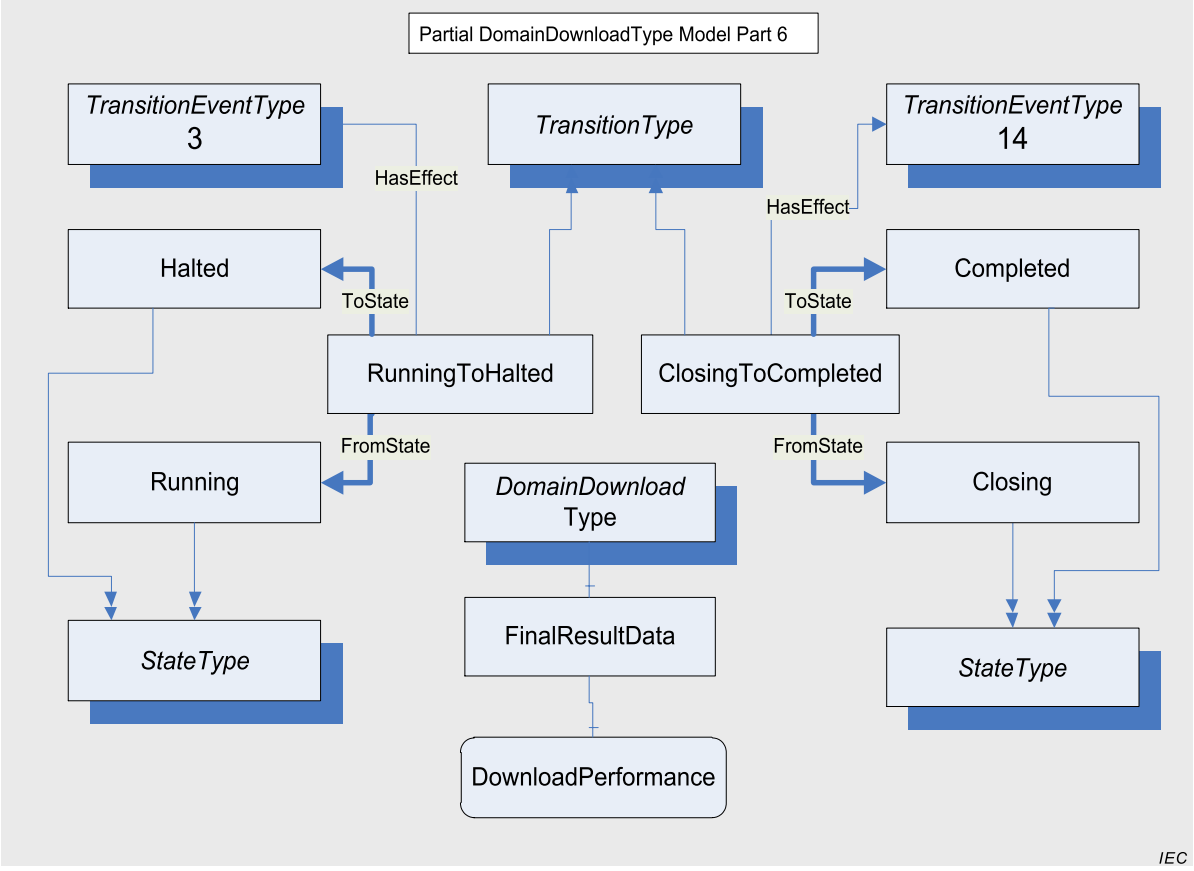

Figure A.10 – Modèle Running To Completed

A.2.6.3 Séquence des opérations

La Figure A.11 illustre une séquence normale d'échanges de service entre un *Client* et le *Serveur* susceptible de se produire au cours de la durée de vie d'une *Invocation de Programme* de DomainDownloadType.

Copyright International Electrotechnical Commission

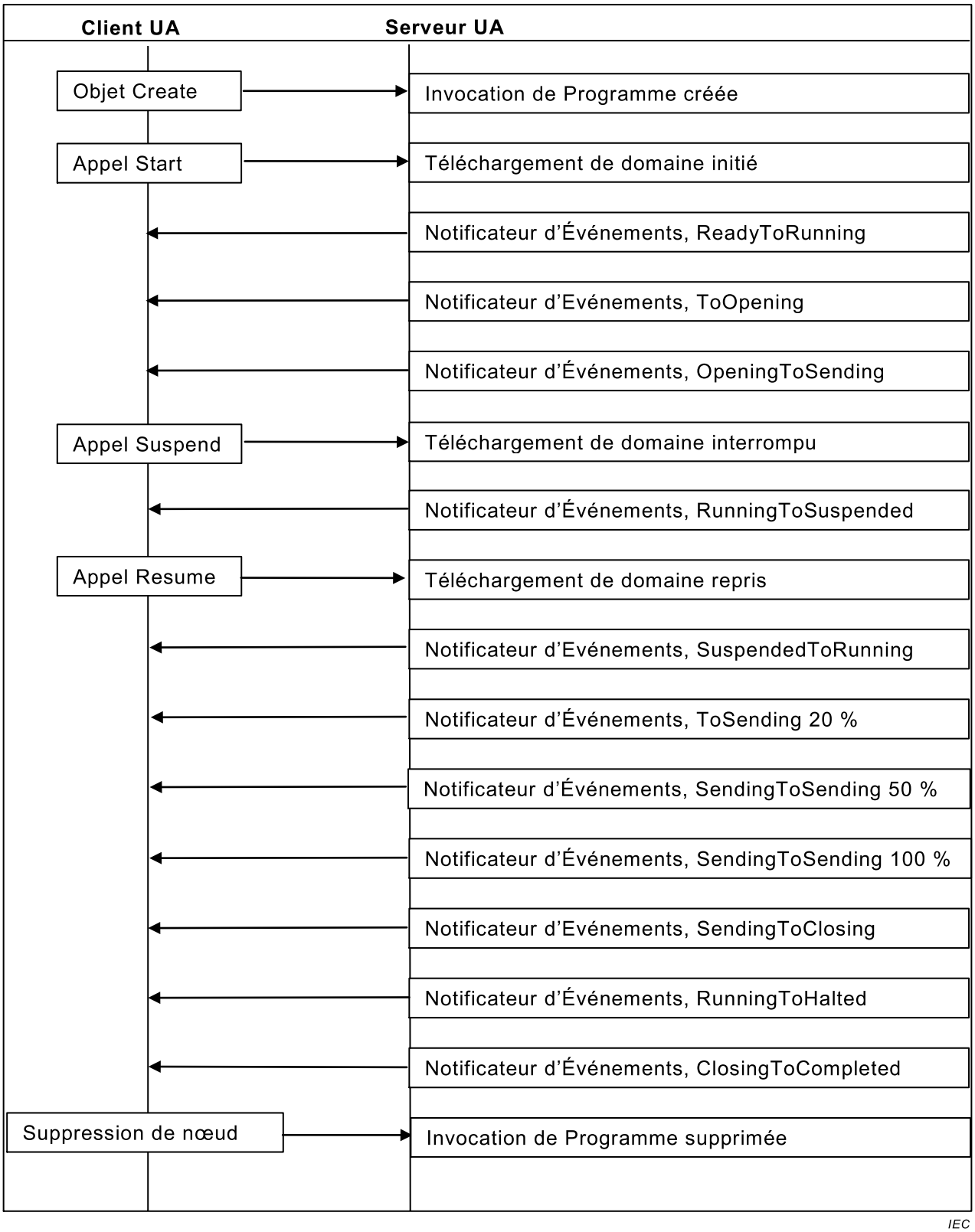

Figure A.11 – Séquence des opérations

Copyright International Electrotechnical Commission

Copyright International Electrotechnical Commission

Copyright International Electrotechnical Commission

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch

Copyright International Electrotechnical Commission