IEC 62541-10

Edition 1.0 2012-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 10: Programs

Architecture unifiée OPC – Partie 10: Programmes

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.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you 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It also gives information on projects, replaced and withdrawn publications.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 on-line 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 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

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 la CEI

La Commission Electrotechnique Internationale (CEI) 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 CEI

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

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI 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.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. 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 au monde 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 les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

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.

IEC 62541-10

Edition 1.0 2012-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 10: Programs

Architecture unifiée OPC – Partie 10: Programmes

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX X

ICS 25.040.40; 25.100.01

ISBN 978-2-83220-284-5

CONTENTS

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.

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

FDIS	Report on voting
65E/244/FDIS	65E/269/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.

INTRODUCTION

This International Standard is a specification intended for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that inter-operate seamlessly together.

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

1 Scope

This part of the IEC 62541 series specifies the standard representation of *Programs* as part of the OPC Unified Architecture and its defined information model. 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 *NodeClasses* and *Attributes* is specified in IEC 62541-3. The services such as those used to invoke the *Methods* used to manage *Programs* 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 62541-3:2010, *OPC unified architecture – Part 3: Address Space Model*

IEC 62541-4:2011, *OPC unified architecture – Part 4: Services*

IEC 62541-5, *OPC unified architecture – Part 5: Information Model*

IEC 62541-7, *OPC unified architecture – Part 7: Profiles*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

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

3.1.1
function
programmatic task performed at 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
ObjectType Node that represents the type definition of a *Program* and is a subtype of the *FiniteStateMachineType*

3.1.4
Program Control Method
Method specified by this specification 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: The Program Invocation is distinguished from other Object instances of the same ProgramType by the object node's unique browse path.

3.2 Abbreviations

API Application Programming Interface
DA Data Access
FSM Finite State Machine
HMI Human Machine Interfaces
PCM Program Control Method
PGM Program
PI Program Invocation
PLC Programmable Logic Controller
UA Unified Architecture
UML Unified Modelling Language

4 Concepts

4.1 General

Integrated automation facilities manage their operations through the exchange of data and coordinated invocation of system functions like 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 of, 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.

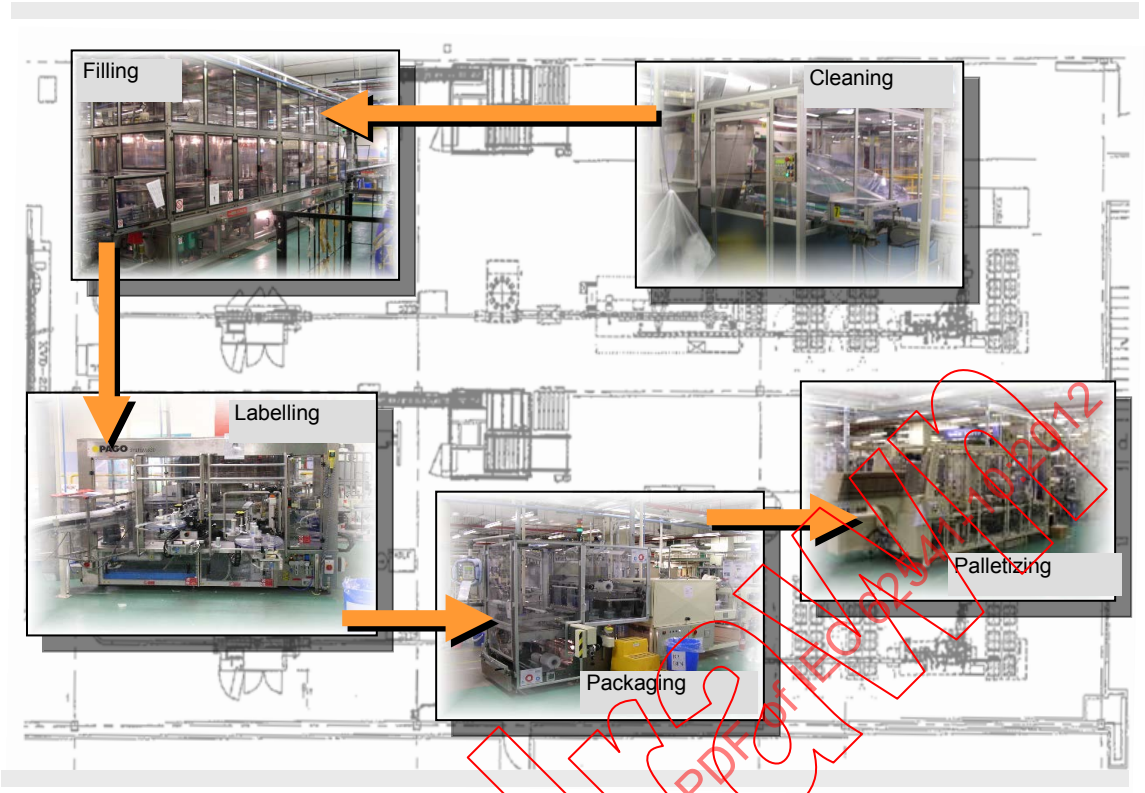


Figure 1 – Automation facility control

Methods and *Programs* model functions typically having 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, 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 specification describes the extensions to, or specific use of the core capabilities defined in IEC 62541-5. Support for the feature described in this specification is described by means of *Profiles* in IEC 62541-7.

4.2 Programs

4.2.1 Overview

Programs are complex functions in a server or underlying system that can be invoked and managed by an OPC *UA 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.

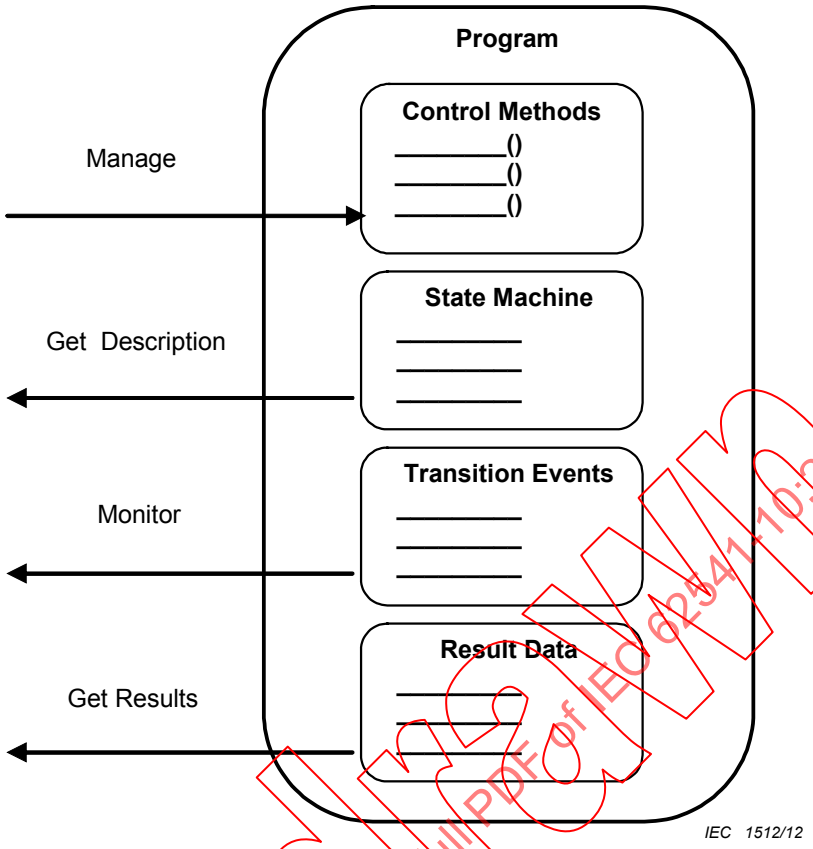


Figure 2 – Program illustration

Programs are state full, transitioning 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 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 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

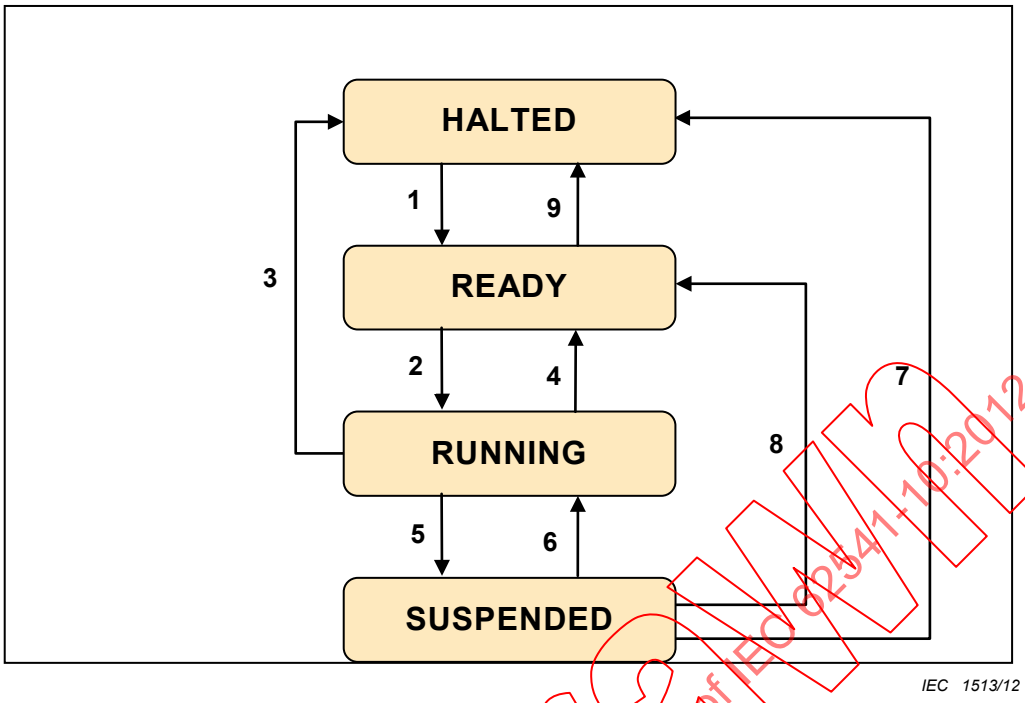


Figure 3 – Program States and Transitions

4.2.3 Program States

A standard set of base states are 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 substates can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions. Standards bodies and industry groups may extend the base *Program Finite State Model* to conform to industry models. For example, the Halted state can include the sub states "Aborted" and "Completed" to indicate if 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.

4.2.4 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

4.2.5 Program State Transition Stimuli

The stimuli or causes for a *Program's* state transitions can be internal to the *Server* or external. Completion of machining steps, the detection of an alarm condition, or the transmission of a data a 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*.

4.2.6 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 action's performed by the *Program*. This specification 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*.

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 62541-4:2011, 5.11 defines a return status. This return status indicates the success of the *Program Control Method* or a reason for its failure.

4.2.7 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.

4.2.8 Program Result Data

4.2.8.1 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.

4.2.8.2 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.

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.

4.2.8.3 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.

4.2.8.4 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.

4.2.9 Program Lifetime

4.2.9.1 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 62541-3:2010, 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, 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 62541-4:2011, 5.7 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.

4.2.9.2 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*.

4.2.9.3 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 set that characterize 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*.

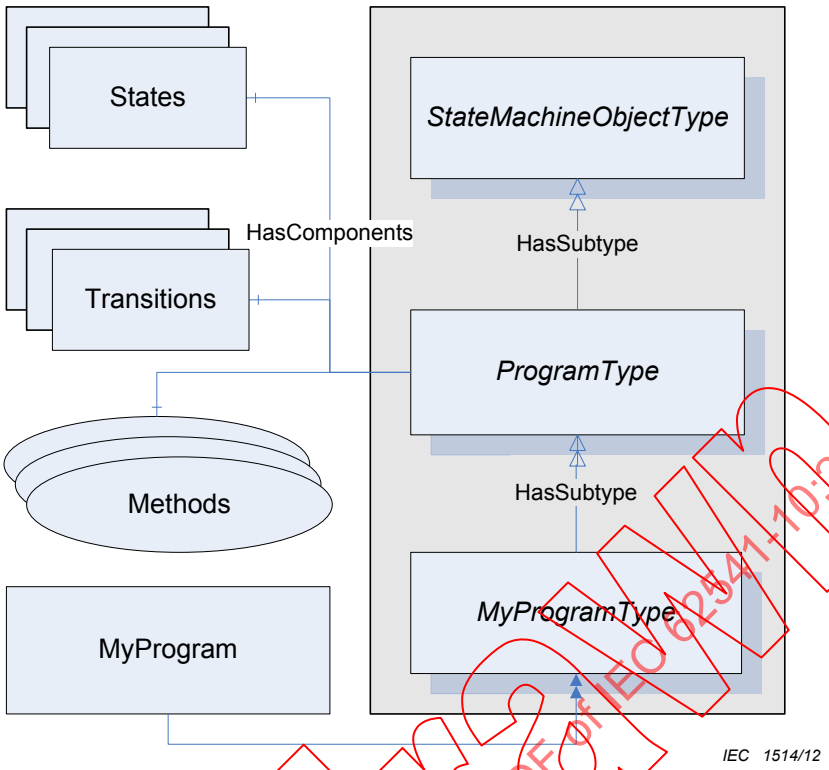


Figure 4 – Program Type

The base *ProgramType* defines the standard *Finite State Machine* specified for all *Programs*. This includes the states, transitions, transition causes (*Methods*) and effects (Events). Sub-types 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*.

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		ProgramDiagnos-ticType	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

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: 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, there is no limit to the number of restarts. If the value is zero, 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, there is no limit.

5.2.3 ProgramType Components

5.2.3.1 Overview

The *ProgramType Components* consists of a set of references to the object instances of *StateTypes*, *TransitionTypes*, *EventTypes* and the *Methods* that collectively define the *Program FiniteStateMachine*.

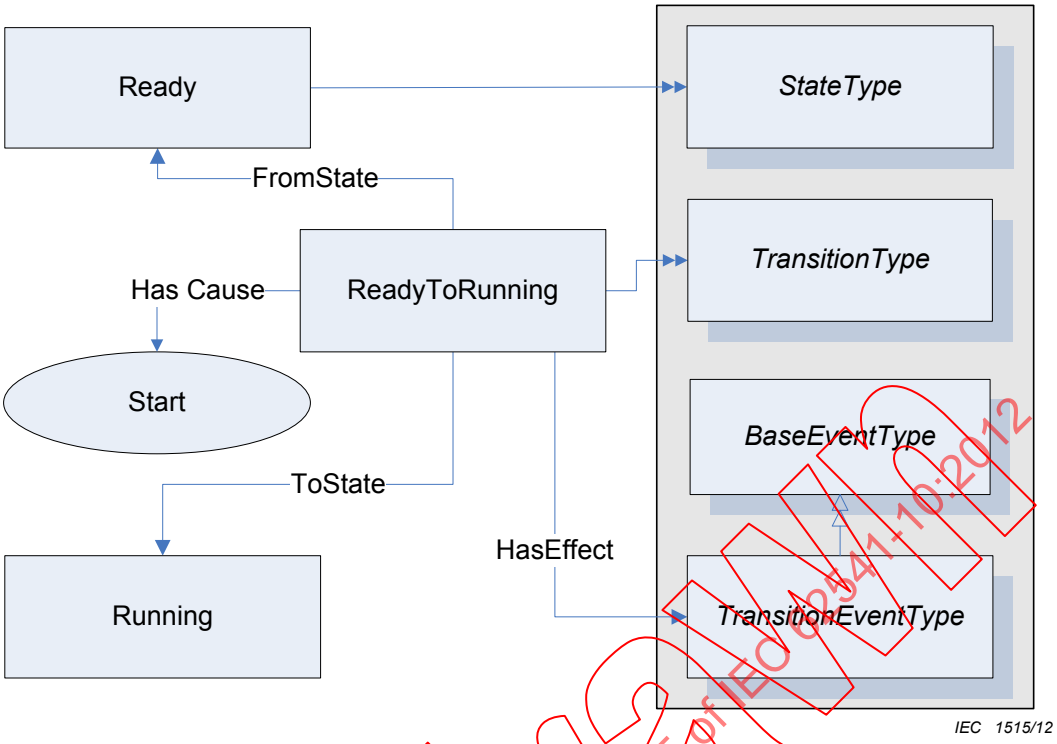


Figure 5 – Program FSM References

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 State *Objects* are instances of the *StateType* defined in IEC 62541-5:2011, 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 *FinitStateMachine*.

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	
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 can not yet 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 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 62541-5:2011, 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 a method and when effects are optional.

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	AuditProgramTransition-EventType			
ReadyToRunning	HasProperty	TransitionNumber	2	PropertyType	
	ToState	Running		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToHalted	HasProperty	TransitionNumber	3	PropertyType	
	ToState	Halted		StateType	
	FromState	Running		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToReady	HasProperty	TransitionNumber	4	PropertyType	
	ToState	Ready		StateType	
	FromState	Runnning		StateType	
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToSus-pended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
SuspendedToRun-ning	HasProperty	TransitionNumber	6	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
SuspendedToHalted	HasProperty	TransitionNumber	7	PropertyType	
	ToState	Halted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
SuspendedToReady	HasProperty	TransitionNumber	8	PropertyType	
	ToState	Ready		StateType	
	FromState	Suspended		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
ReadyToHalted	HasProperty	TransitionNumber	9	PropertyType	
	ToState	Halted		StateType	
	FromState	Ready		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			

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.*

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 type *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*.

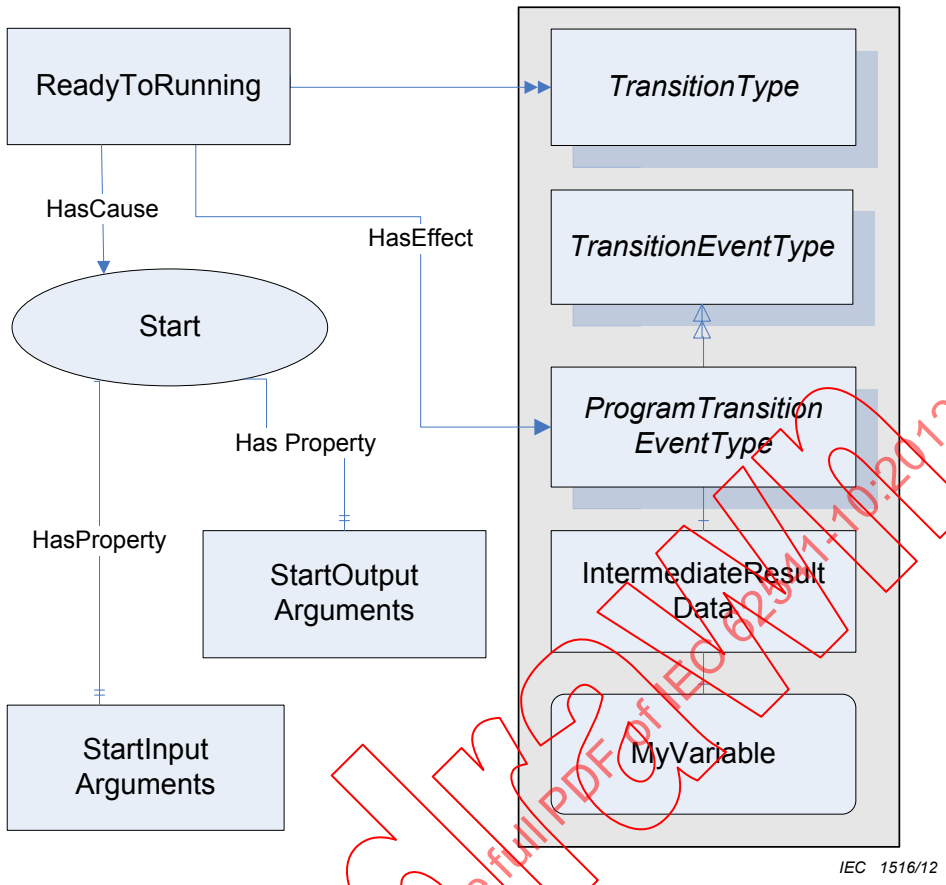


Figure 6 – ProgramType Causes and Effects

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,* omit the *Causes* reference to that *Method* from the *ProgramType* references. If a *Method's Causes* reference is omitted from the *ProgramType*, 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 *ProgramTransitions*.

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

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
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 *CurrentState* 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 *CurrentState* 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.

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 program type. 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**	**Model-lingRule**
Inherits the *Properties* of the base *TransitionEventType* defined in IEC 62541-5:2011, B.4.16.					
HasComponent	Object	Intermedia-teResult		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*.

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	

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
	HasProperty	ToStateNumber	1	PropertyType	
	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**
Inherits the *Properties* of the *AuditUpdateStateEventType* defined in IEC 62541-5:2011, Annex B.4.17.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateStateEventType* defined in IEC 62541-5:2011, B.4.17 except as noted below IEC 62541-5.

The *Status Property,* specified in IEC 62541-5:2011, 6.4.3 "AuditEventType", 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 62541-5:2011, 6.4.2 "BaseEventType" identifies the *Method* causing the *ProgramTransition* 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 62541-5:2011, 6.4.3 "AuditEventType", identifies the user of the *Client* that issued the *Program Control Method* if it is associated with this *Program State Transition*.

The *ActionTimeStamp Property,* specified in IEC 62541-5:2011, 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 FinalResultData.

5.2.8 ProgramDiagnosticType

The *ProgramDiagnoticType* provides information that can be used to aid in the diagnosis of *Program* problems. This object contains a collection of *Variables* that chronicle the *ProgramInvocation's* activity. Table 12 specifies the *Variables* that compose the *ProgramDiagnoticType*.

Table 12 – ProgramDiagnosticType

Attribute	Value			
BrowseName	ProgramDiagnosticsType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType 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	Int32	Mandatory
HasComponent	Variable	LastMethodInputArguments	InputArguments	Mandatory
HasComponent	Variable	LastMethodOutputArguments	OutputArguments	Mandatory
HasComponent	Variable	LastMethodCallTime	UTCTime	Mandatory
HasComponent	Variable	LastMethodReturnStatus	returnStatus	Mandatory

The *CreateSessionId* contains the *SessionId* of the session on which the call to the *Create Method* was issued to create the *Program Invocation*.

The *CreateClientName* is the name of the client of the session that created the *Program Invocation*.

The *InvocationCreationTime* identifies the time the *Program Invocation* was created.

The *LastTransitionTime* identifies the time of the last program state transition that occurred.

The *LastMethodCall* identifies the last *Program Method* called on the *Program Invocation*.

The *LastMethodSessionId* contains the *SessionId* of the session on which the last Program Control Method call to the *Program Invocation* was issued.

The *LastMethodClientName* is the name of the client of the session that made the last Method call to the *Program Invocation*.

The *LastMethodInputArguments* preserves the values of the input arguments on the last *Program Method* call.

The *LastMethodOutputArguments* preserves the values of the output arguments on the last *Program Method* call.

The *LastMethodCallTime* identifies the time of the last Method call to the *Program Invocation*.

The *LastMethodReturnStatus* preserves the value of the *returnStatus* for the last *Program Control Method* requested for this *Program Invocation.*

Annex A
(informative)

Program Example

A.1 Overview

This example illustrates the use of an OPC UA *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.

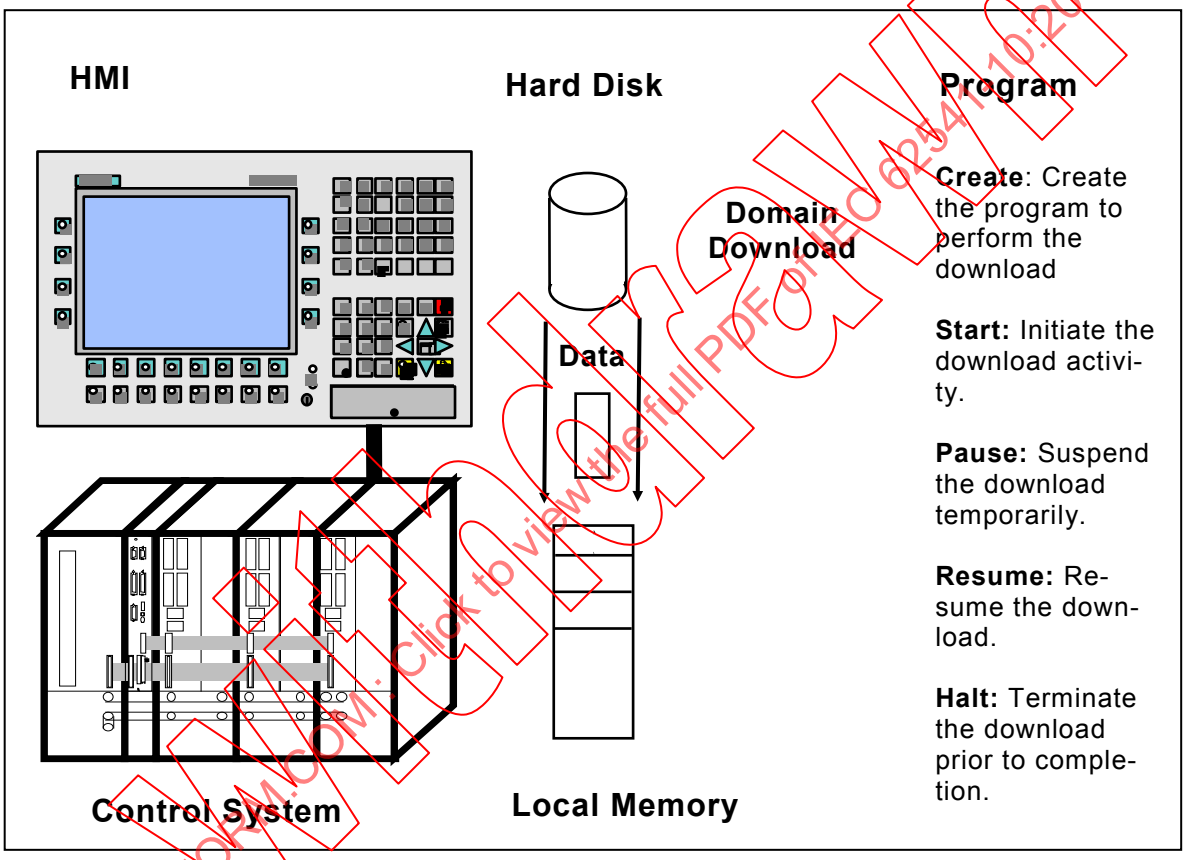


Figure A.1 – Program Example

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, performance information is persisted at the OPC *UA Server*.

A.2 DomainDownload Program

A.2.1 General

The OPC *UA Client* uses the **"DomainDownload"** *Program* to manage and monitor the download of a domain at the OPC UA *Server*.

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 OPC *UA 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 OPC UA *Client* to monitor the activity of the download at a finer resolution.

An instance (Program Invocation) of a DownloadDomain *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.

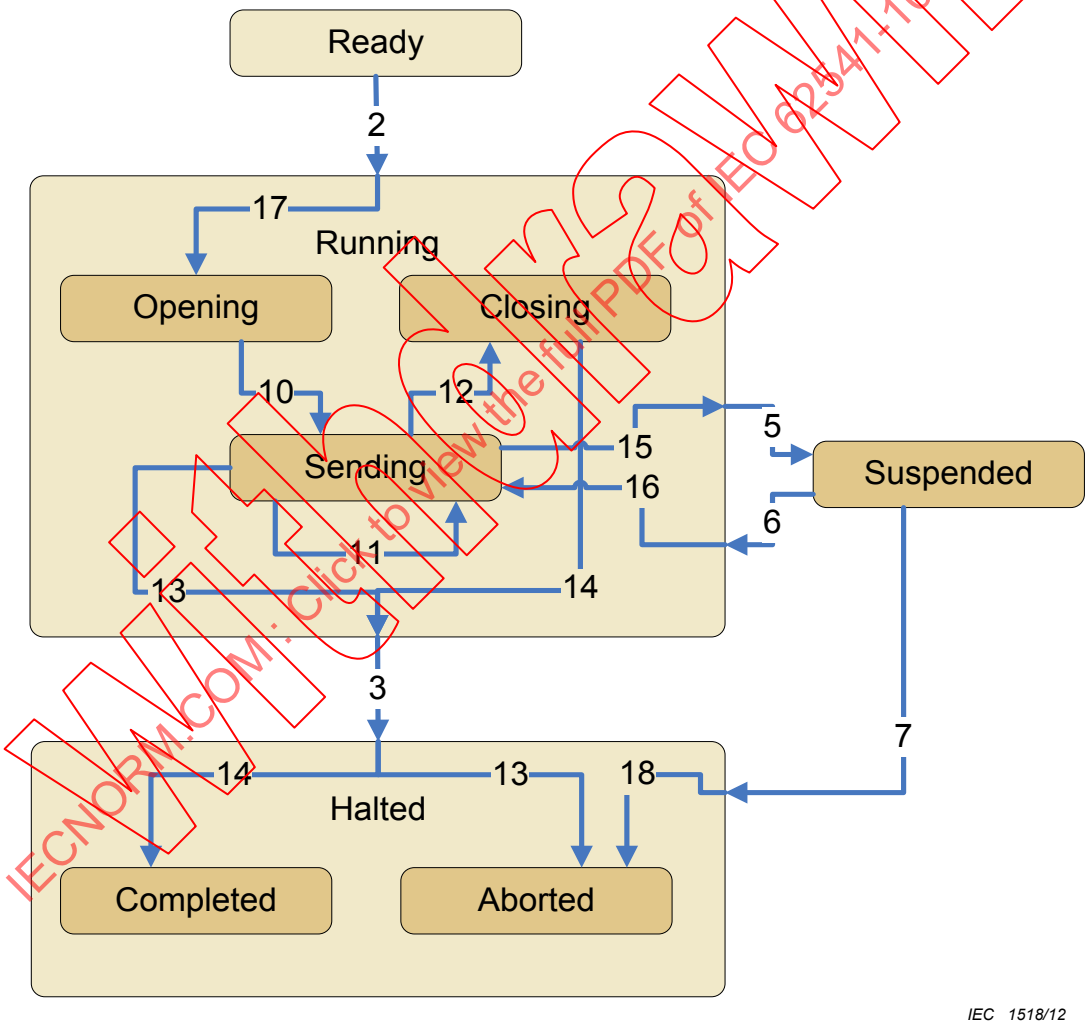


Figure A.2 – DomainDownload State Diagram

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, the download is halted, and the Program transitions to the Aborted state, otherwise the programs halts in the Completed state. The states are presented in Table A.1 along with the state transitions.

A.2.3 DomainDownload Transitions

The valid state transitions specified for the DomainDownload *Program* and 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 distinct 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 the sub-state change. For example, ReadytoRunning and ToOpening 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 OPC *UA 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.

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 event types trigger a specific event notification to the OPC *UA 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 OPC *UA Client* along with the notification. Each time the transition occurs, data items describing the amount and percentage of data transferred is sent to the OPC *UA 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 an OPC *UA Client* to interpret or validate the actions of the Program.

A.2.6.2 DomainDownloadType

The DomainDownloadType is a subtype derived from the OPC *UA 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.

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		ProgramDiagnos-ticType	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 DomianDownload Program Type. This State Machine Type definition identifies the State types that compose the sub states for the Program's Running State type.

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.

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 State Machine Type definition identifies the State types that compose the sub states for the Program's Halted State type.

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 State Types associated with the Finish State Machine Type. Note these are final states and that they have no associated transitions between them.

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 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	Deletabe	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 OPC *UA Client*. The *Program Invocation* will not delete itself when halted, but will persist until explicitly removed by the OPC *UA Client*. A DomainDownload *Program Invocation* cannot be reset to restart. The OPC *UA Server* will support up to 500 concurrent DomainDownload *Program Invocations*.

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.

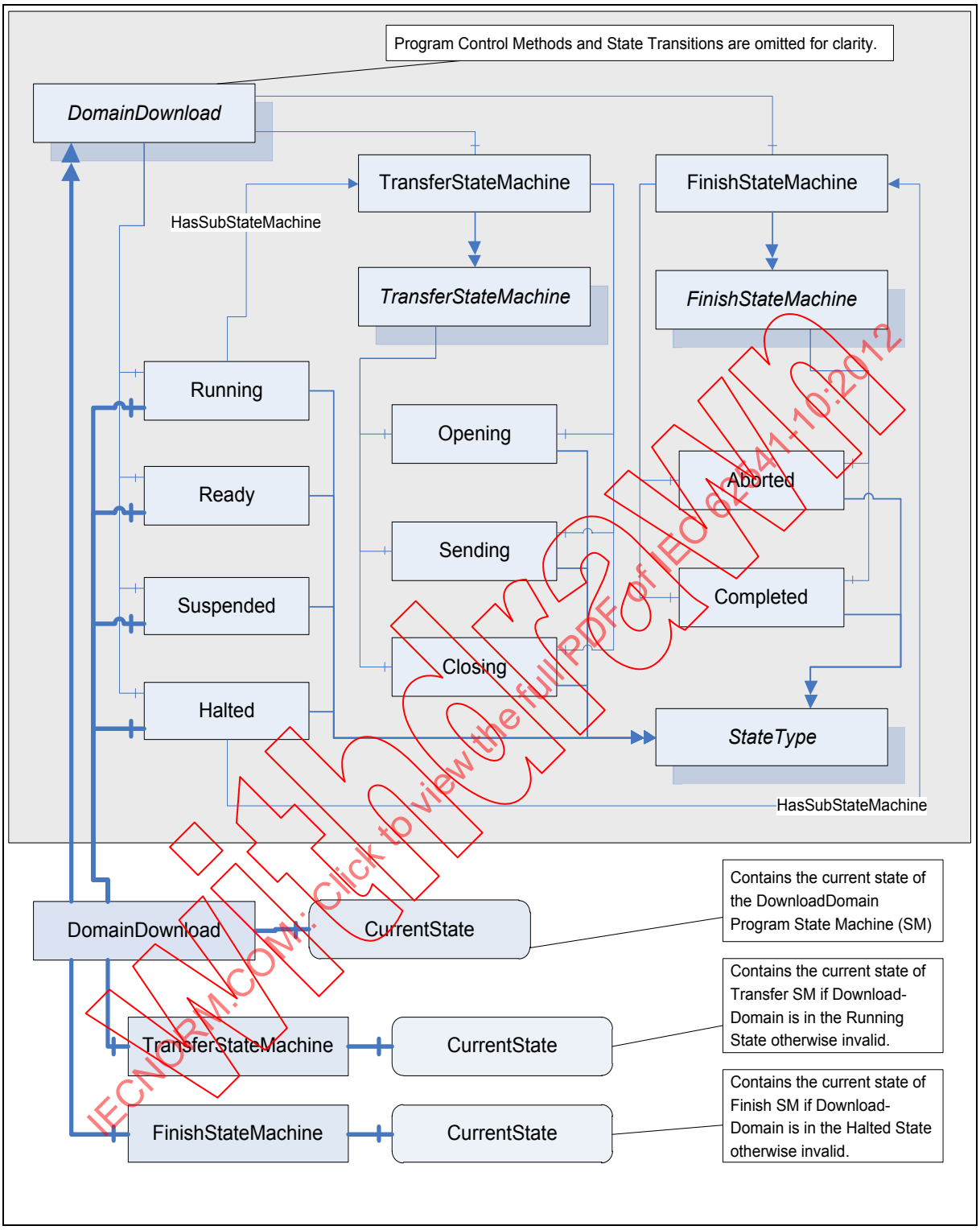


Figure A.3 – DomainDownloadType Partial State Model

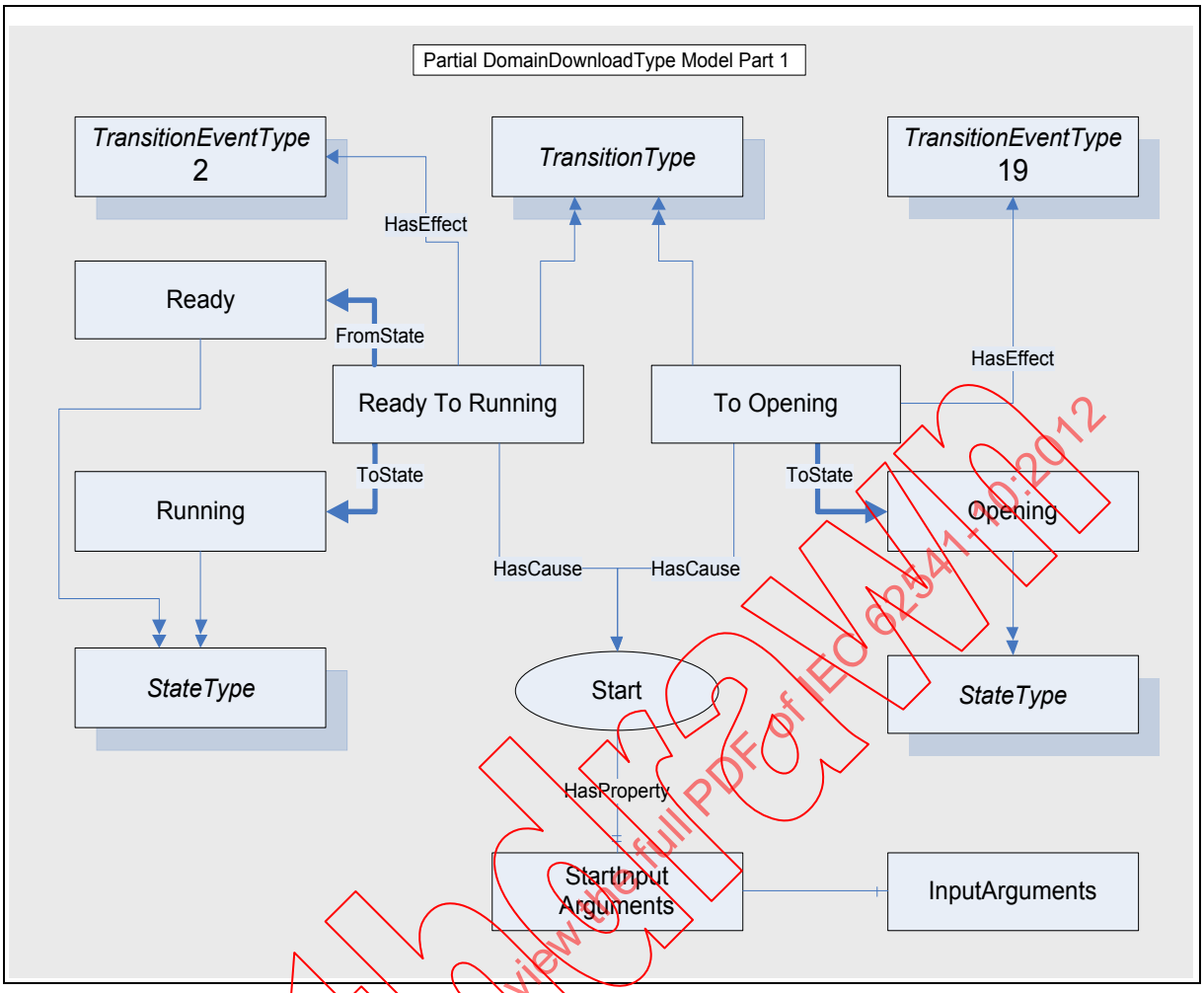
Table A.8 specifies the *ProgramTransitionTypes* that are defined in addition to the OPC *UA ProgramTransitonTypes* specified for *Programs* in Table 7. These types associate the Transfer and Finish sub state machine states with the states of the base *Program*.

Table A.8 – Additonal 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			
ClosingToComplet-ed	HasProperty	TransitionNumber	14	PropertyType	
	ToState	Completed		StateType	
	FromState	Closing		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToSus-pended	HasProperty	TransitionNumber	15	PropertyType	
	ToState	Suspended		StateType	
	FromState	Sending		StateType	
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransitionEventType			
SuspendedToSend-ing	HasProperty	TransitionNumber	16	PropertyType	
	ToState	Sending		StateType	
	FromState	Suspended		StateType	

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 States, Methods, Transitions and EventTypes are identified for one or two state transitions.



IEC 1520/12

Figure A.4 – ReadyToRunning 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 OPC *UA 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 OPC *UA 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.

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. Event notifiers are generated for each state transition (10-12), when they occur.

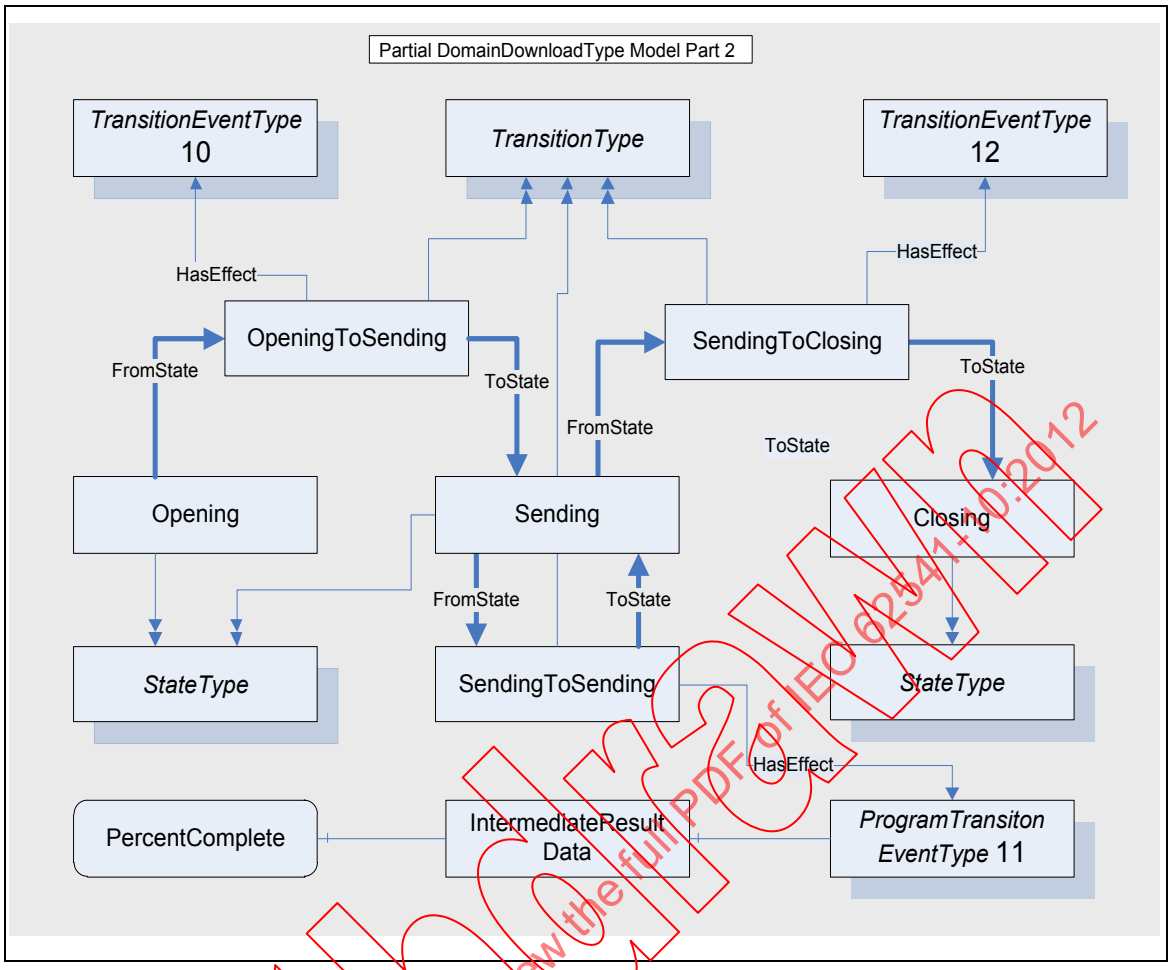


Figure A.5 – OpeningToSending ToClosing Model

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 OPC *UA Client* with the event notification. Table A.11 defines the *IntermediateResults ObjectType* and Table A.13 defines the variables of the *ObjectType*.

Table A.11 – Intermediate Results 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

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

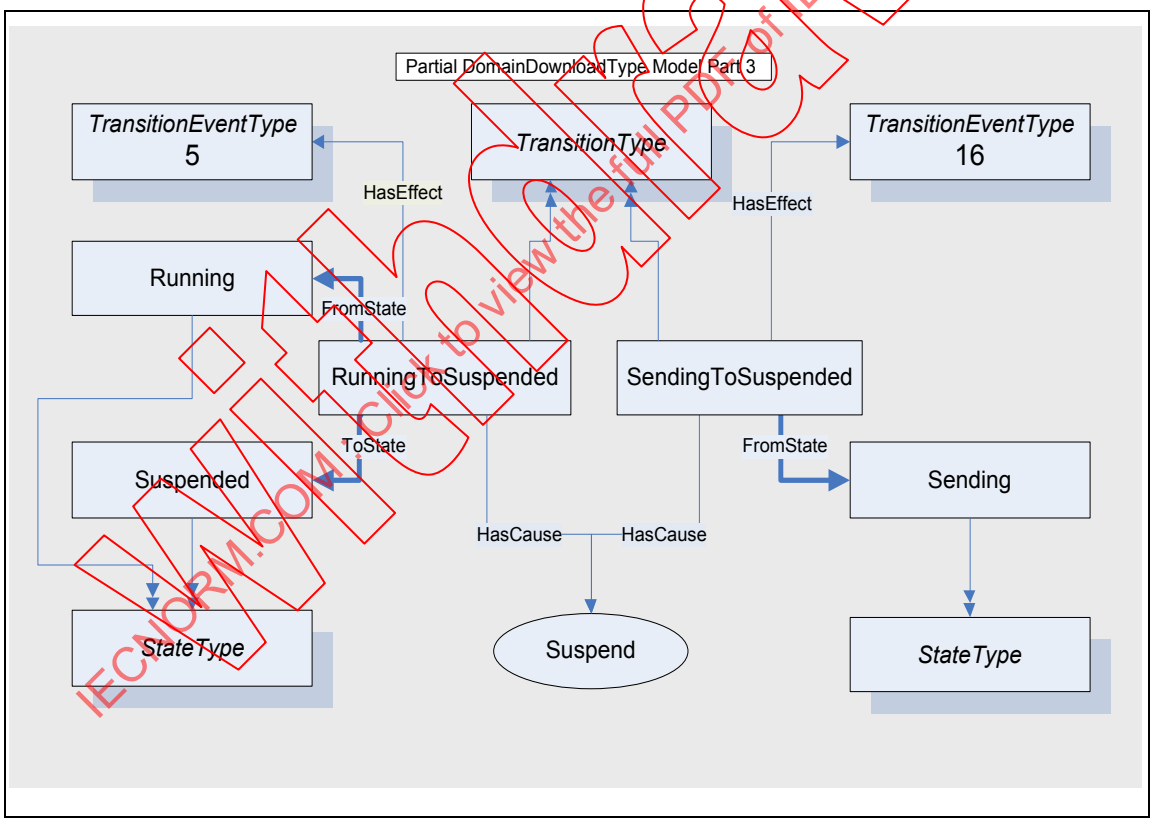


Figure A.6 – RunningToSuspended Model

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

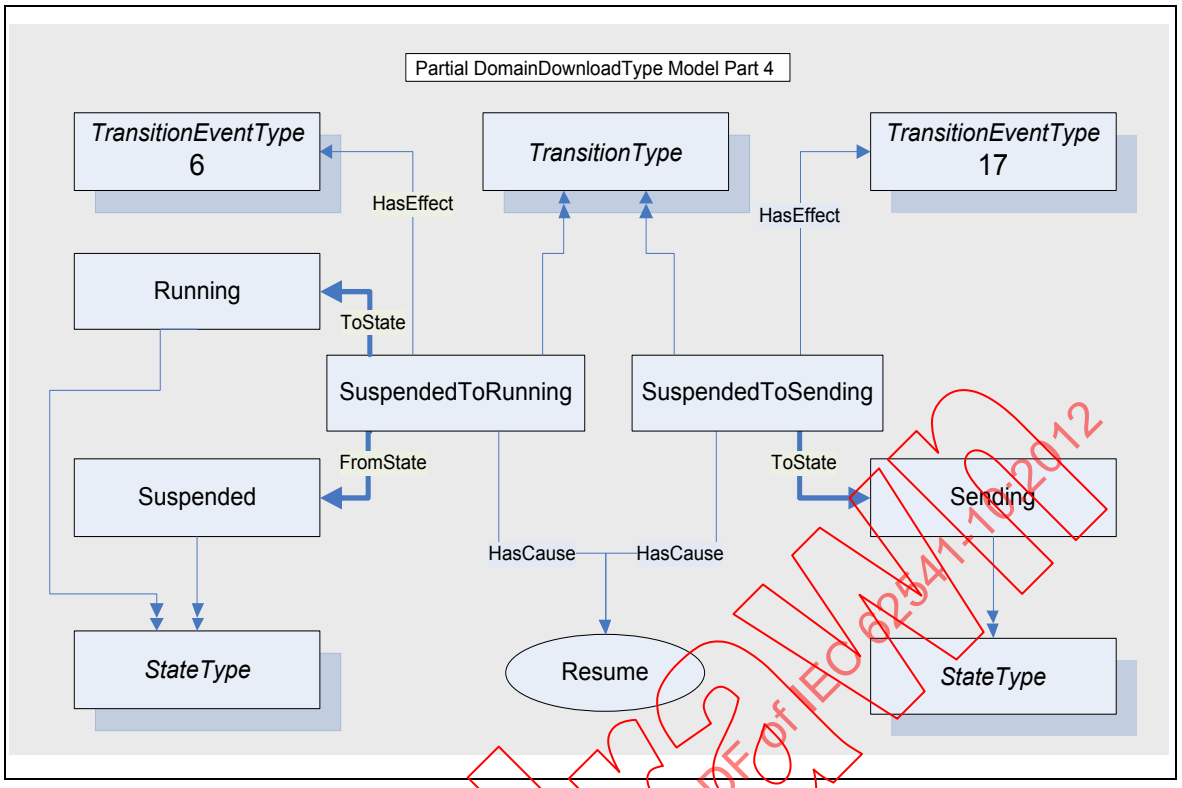


Figure A.7 – SuspendedToRunning Model

The model for the RunningToHalted 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 OPC *UA Client* can terminate the download of domain data to the control. The transition from Running to Halted evokes the event notifiers for TransitionEventTypes 3 and 15. The TransitionEventType 15 indicates the transition from the Sending State as the Running State is exited and to the Aborted State as the Halted State is entered.

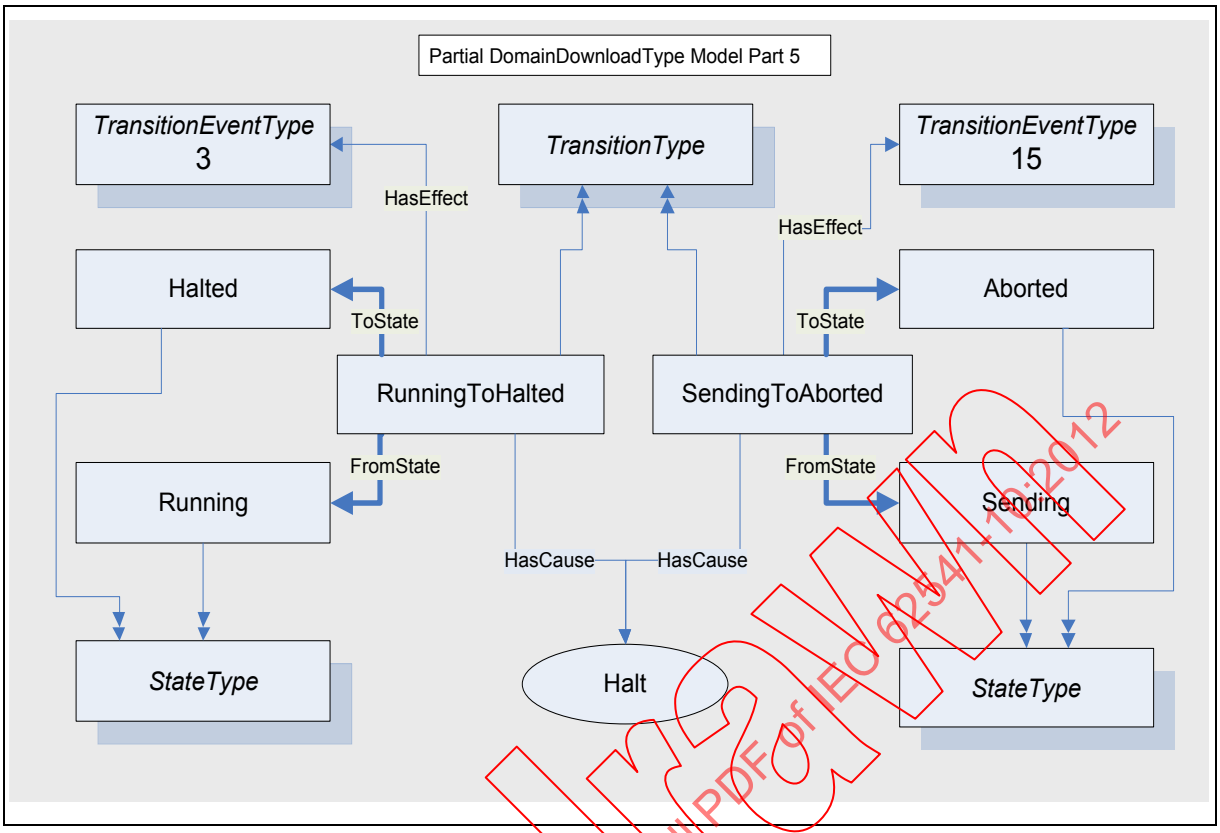


Figure A.8 – RunningToHalted – Aborted Model

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

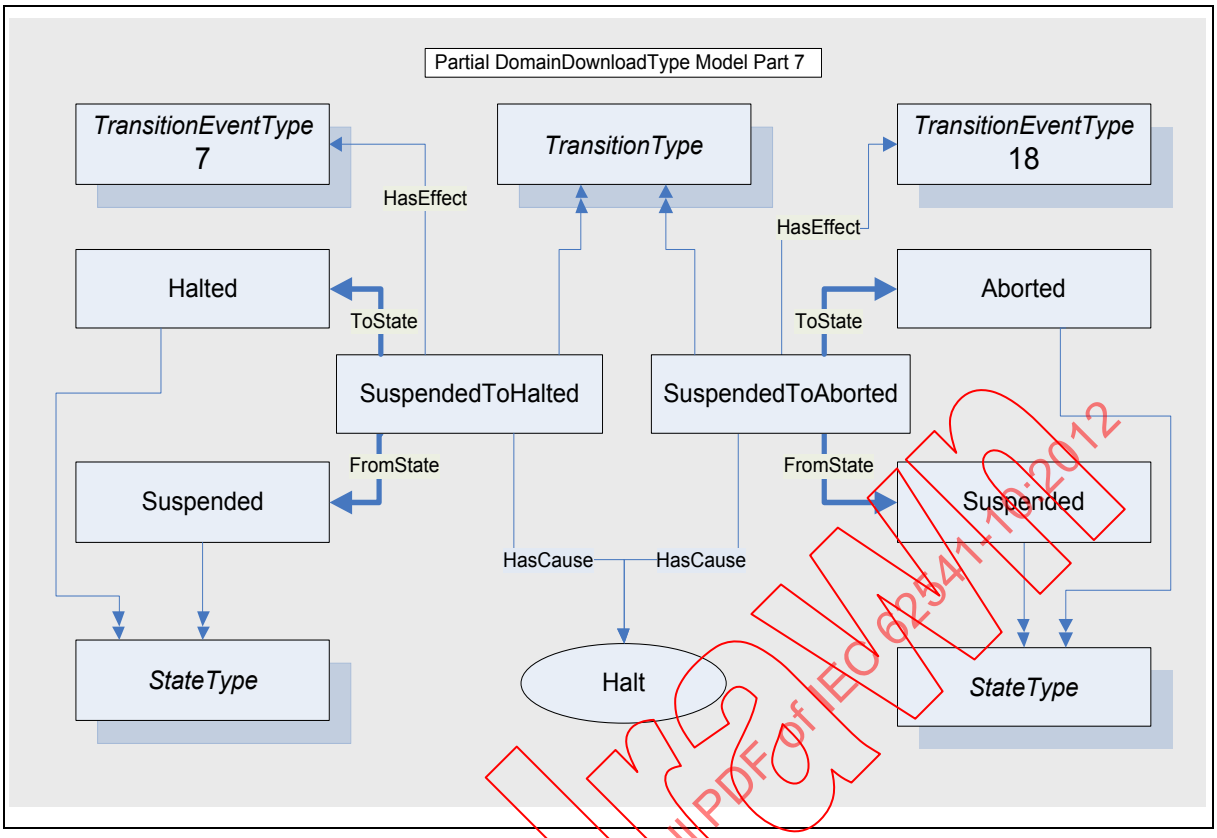


Figure A.9 – SuspendedToAborted Model

IEC 1525/12

The model for the RunningToCompleted 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 evokes the event notifiers for TransitionEventTypes 3 and 14. The TransitionEventType 14 indicates the transition from the Closing State as the Running State is exited and 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 OPC UA Clients who are not subscribed to event notifications. The result data is described in Table A.13 and the variables in Table A.14.

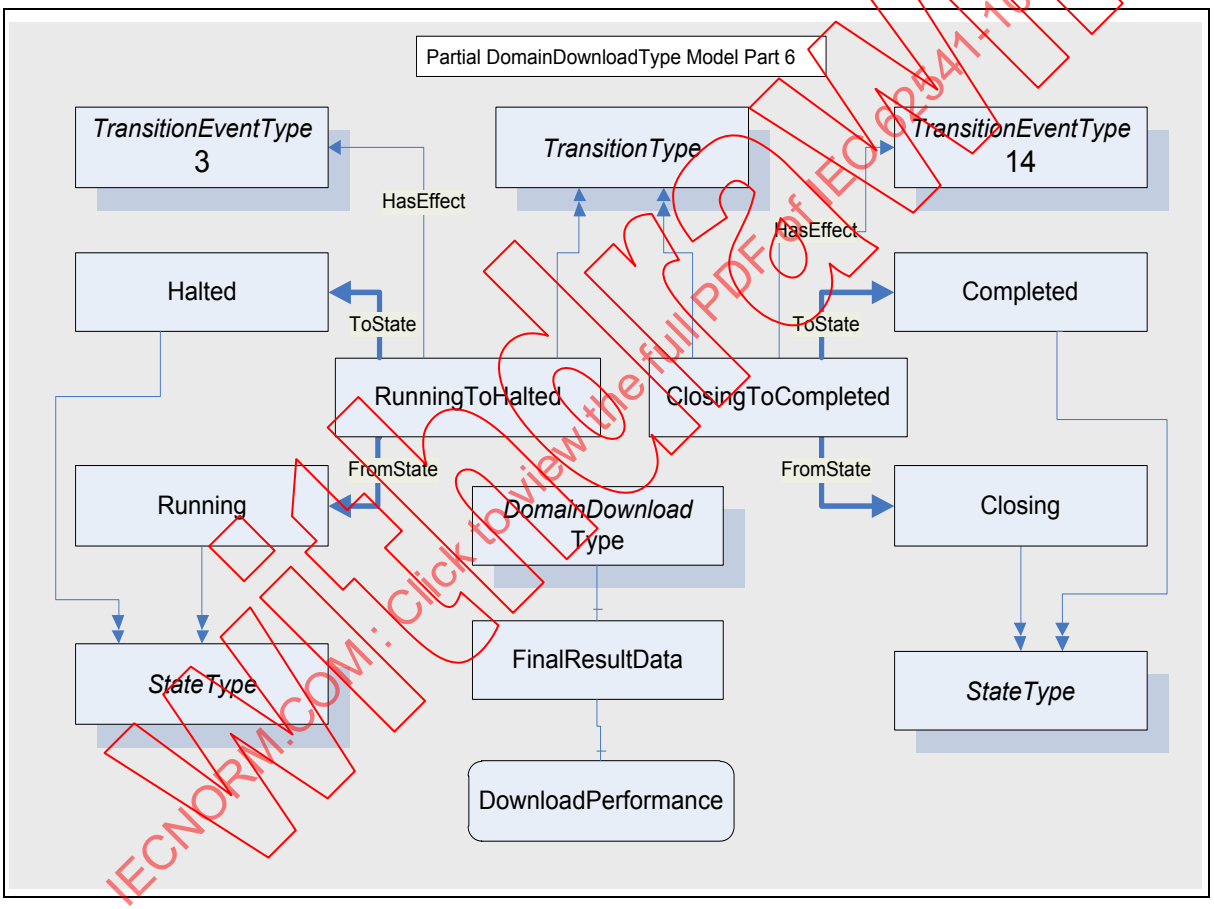
Table A.13 – Final Result Data

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.

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.



IEC 1526/12

Figure A.10 – RunningToCompleted Model

A.2.6.3 Sequence of Operations

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

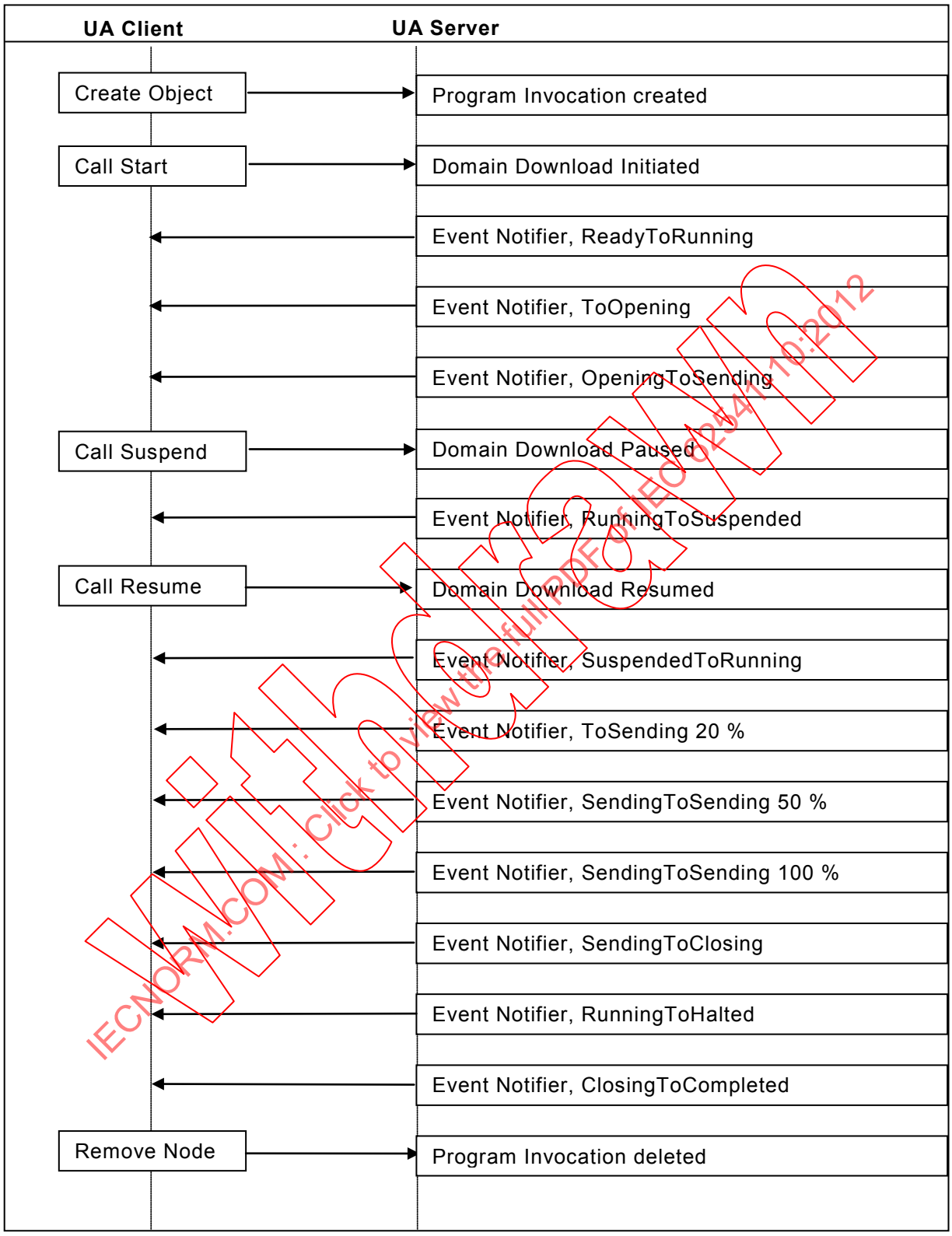


Figure A.11 – Sequence of Operations

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

ARCHITECTURE UNIFIEE OPC –

Partie 10: Programmes

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI 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, la CEI – 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 la CEI"). 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 la CEI, participent également aux travaux. La CEI 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 la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI 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 la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI 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 la CEI. La CEI 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 la CEI, 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 la CEI, 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 la CEI ou de toute autre Publication de la CEI, 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 la CEI peuvent faire l'objet de droits de brevet. La CEI 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 CEI 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 la CEI: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/244/FDIS	65E/269/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/CEI, Partie 2.

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

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 la CEI 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.

INTRODUCTION

La présente Norme internationale est une spécification destinée aux développeurs d'applications OPC UA. La spécification est le résultat d'un processus d'analyse et de conception pour développer une interface normalisée qui facilite l'interopérabilité d'applications issues de divers fournisseurs.

ARCHITECTURE UNIFIEE OPC –

Partie 10: Programmes

1 Domaine d'application

La présente partie de la série CEI 62541 spécifie la représentation normalisée des *Programmes* faisant partie intégrante de l'Architecture Unifiée OPC et son modèle d'information défini. Elle comprend la description des *Classes de Nœuds,* et des *Propriétés, Méthodes* et *Évènements* normalisés, ainsi que les paramètres associés relatifs au comportement et aux informations applicables aux *Programmes*.

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

CEI/TR 62541-1:2010, *OPC Unified Architecture – Part 1: Overview and Concepts* (disponible uniquement en anglais)

CEI 62541-3:2010, *OPC unified architecture – Part 3:Address Space Model* (disponible uniquement en anglais)

CEI 62541-4:2011, *Architecture unifiée OPC – Partie 4: Services*

CEI 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'Information*

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

3 Termes, définitions et abréviations

3.1 Termes et définitions

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

3.1.1
fonction
Function
tâche de programmation assurée au niveau d'un serveur ou d'un dispositif, généralement réalisée par l'exécution de code machine

3.1.2
automate d'états 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 d'états ou de stades discrets

3.1.3
type de programme
ProgramType
ObjectType Node (Nœud de Type d'Objet) qui représente la définition de type de *Programme* et constitue un sous-type du *FiniteStateMachineType* (Type d'automate d'états finis)

3.1.4
méthode de commande de programme
Program Control Method
méthode définie par la présente spécification et 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 d'exploration unique du nœud d'objet.

3.2 Abréviations

API Interface de programme d'application (Application Programming Interface)
DA Accès aux Données (Data Access)
FSM Automate d'états finis (Finite State Machine)
IHM Interfaces Homme-Machine
PCM Méthode de Commande de Programme (Program Control Method)
PGM Programme
PI Invocation de Programme (Program Invocation)
PLC Automate programmable (Programmable Logic Controller)
UA Architecture unifiée (Unified Architecture)
UML Langage de modélisation unifié (Unified Modeling Language)

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.

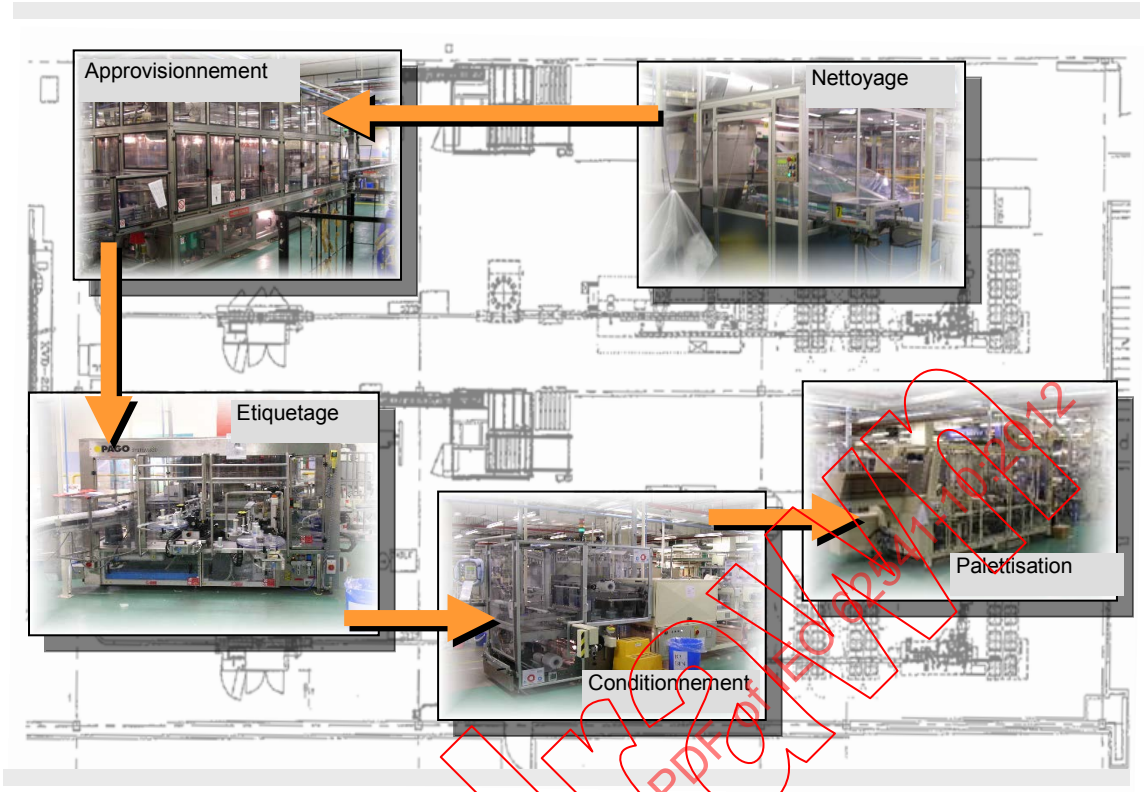


Figure 1 – Commande d'installation d'automatisation

Les fonctions du modèle de *Méthodes* et *Programmes* mises en œuvre dans les *Serveurs OPC* et les systèmes sous-jacents sont généralement régis 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 la CEI 62541-3 et dans la CEI 62541-4.

La présente spécification décrit les extensions aux capacités fondamentales ou leur utilisation particulière requises pour les *Programmes*, telles que définies dans la CEI 62541-5. Les éléments de prise en charge du paramètre caractéristique décrit dans la présente spécification sont décrits dans la CEI 62541-7.

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* OPC *UA* peut appeler et gérer. Les *Programmes* représentent n'importe quel niveau de fonctionnalité au sein d'un système ou d'un processus dont le déroulement doit être contrôlé et qui nécessite la commande ou l'intervention du client. La Figure 2 illustre le modèle.

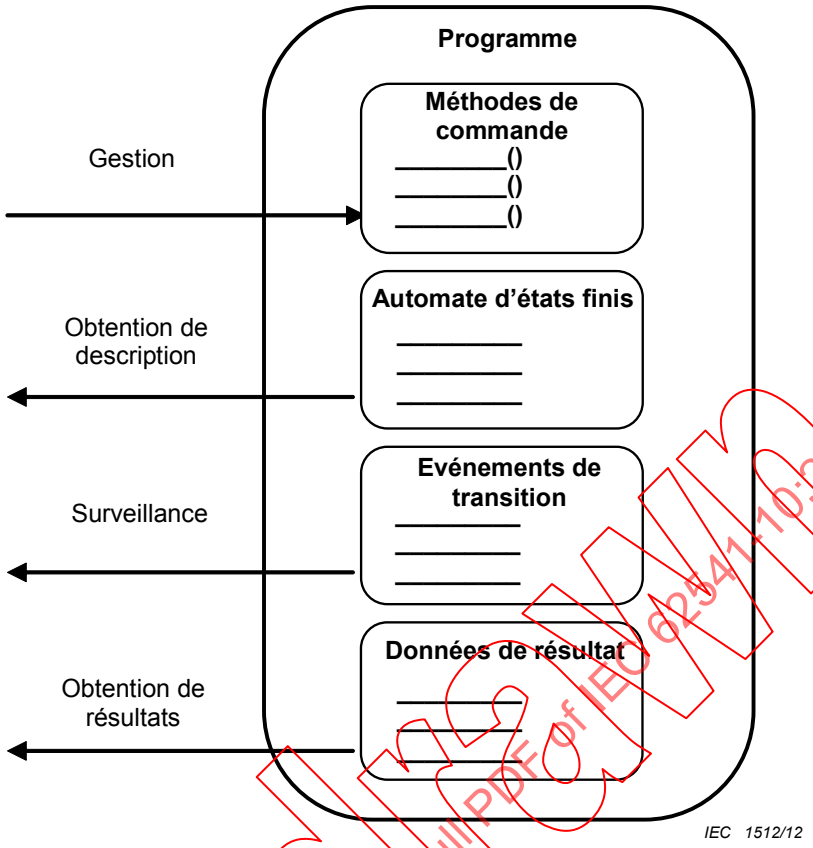


Figure 2 – Illustration d'un programme

L'exécution des *Programmes* se déroule selon une séquence définie d'états. Le comportement des *Programmes* est défini par un *Automate d'Etats Finis de Programme* (*PFSM*)[1]. 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 sont définis.

4.2.2 Automate d'états finis d'un programme

Le Tableau 1 répertorie les états, transitions, causes et effets constitutifs de l'Automate d'Etats Finis d'un Programme qui est illustré à la Figure 3.

1 *PFSM = Program Finite State Machine.*

Tableau 1 – Automate d'états finis d'un 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

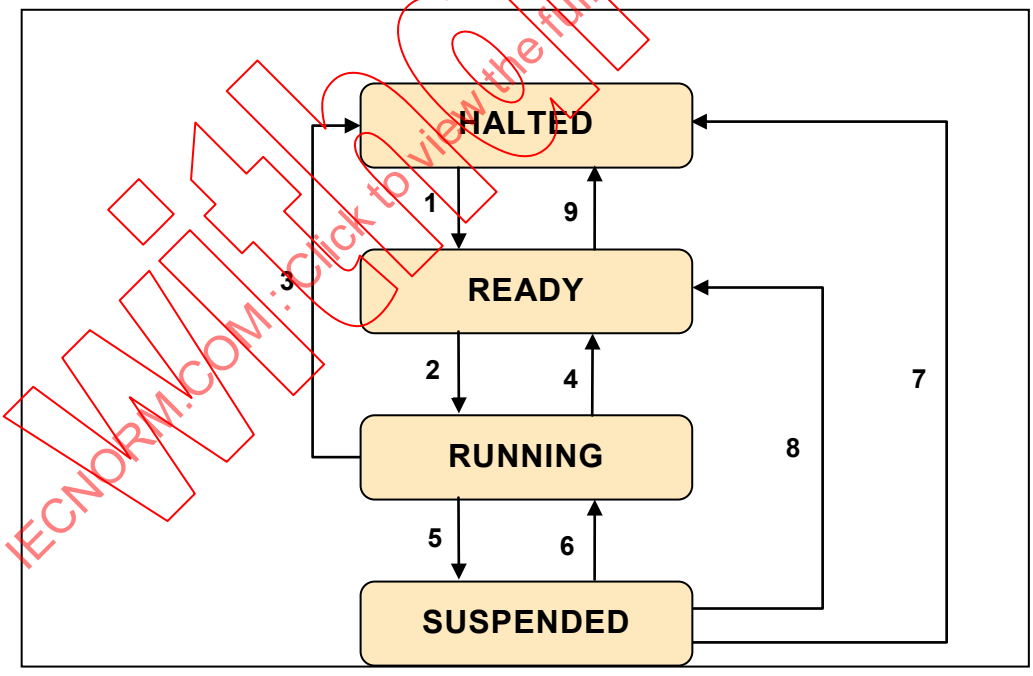


Figure 3 – États et transitions d'un programme

4.2.3 États d'un programme

Un ensemble normalisé d'états de base est défini pour les *Programmes* faisant partie intégrante de l'*Automate d'Etats Finis d'un 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.

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 du processus et décrivent la cause et les effets des stimuli et transitions supplémentaires. Les organismes de normalisation et les groupes industriels peuvent étendre le *Modèle d'Etats Finis de Programme* de base pour se conformer aux 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.

4.2.4 Transitions d'états

Un ensemble normalisé de transitions d'états est défini pour l'*Automate d'Etats 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

4.2.5 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 normalisées sont définies pour agir comme des stimuli pour la commande d'un *Programme*.

4.2.6 Méthodes de commande d'un 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 spécification définit un ensemble de *Méthodes de Commande de Programme* normalisées. Ces *Méthodes* fournissent un nombre suffisant d'informations au client pour exécuter un *Programme*.

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 de l'*Automate d'Etats Finis du Programme* de base.

Tableau 4 – Méthodes de commande d'un 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 au 5.11 de la CEI 62541-4:2011 définit un état de retour ou de réponse. Cet état de retour indique la réussite de la *Méthode de Commande du Programme* ou la raison de son échec.

4.2.7 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'événement. Chaque transition d'états d'un *Programme* est associée à un événement unique. Ces événements 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.

4.2.8 Données résultat d'un programme

4.2.8.1 Vue d'ensemble

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

4.2.8.2 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.

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 de résultats intermédiaires. 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 événements de transition d'états d'un *Programme*. Les événements spécifient les éléments de données pour chaque transition. Lorsque la transition se produit, l'événement 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.

4.2.8.3 Données de résultat finales

Les données de résultat finales 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'événement 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 du Programme.

4.2.8.4 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 événements permettent de conserver l'enregistrement des clients qui interagissent avec un Programme et les transitions d'états du Programme résultants de cette interaction.

4.2.9 Durée de vie d'un programme

4.2.9.1 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 au 5.5.4 de la CEI 62541-3:2010 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 nœud)* défini au 5.7 de la CEI 62541-4:2011 permet de retirer l'Instance du *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.

4.2.9.2 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 décou-

vrir 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*.

4.2.9.3 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 d'Automate d'Etats Finis) et le Modèle de base *ObjectType* (Type d'Objet) présentés dans la CEI 62541-5. Chaque *Programme* comporte une définition de type qui constitue le sous-type du *FiniteStateMachineType*. Le *Type de Programme* décrit le modèle *Automate d'états finis* pris en charge par toute *Invocation d'un 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*.

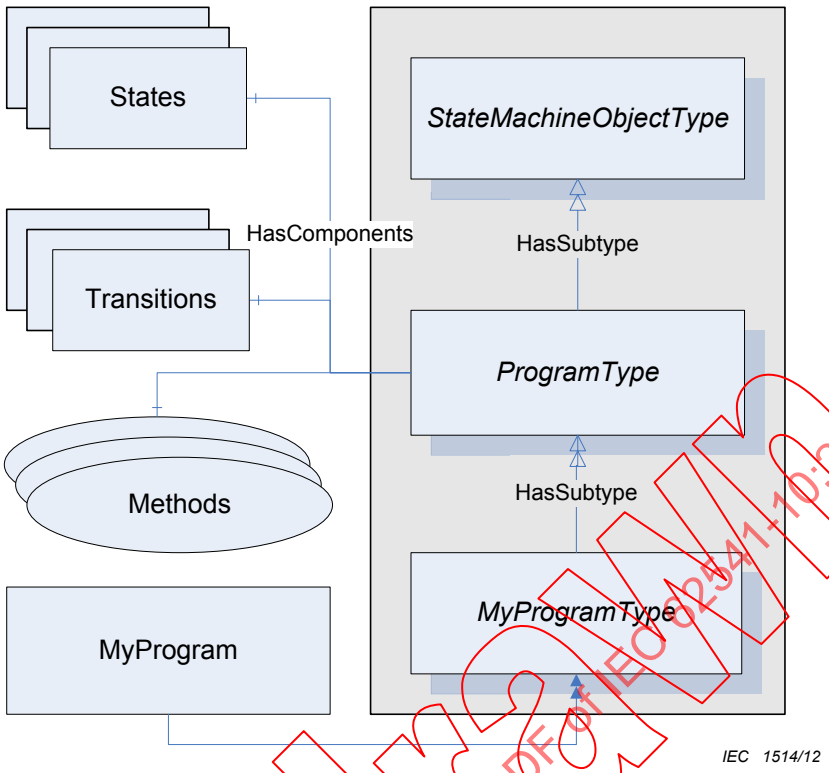


Figure 4 – Type de programme

Le *Type de Programme* de base définit l'*Automate d'états finis* normalisé spécifié pour tous les *Programmes*. Ceci comprend les états, les transitions ainsi que les causes (*Méthodes*) et les effets (Evénements) 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 d'exploration**	**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		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
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 type d'un programme

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

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

La *Propriété AutoDelete* (Suppression Automatique) est un booléen qui indique si les Invocations d'un Programme de ce *Type de Programme* sont retirées par le *Serveur* à la fin de l'exécution. Si la valeur est Faux, ces *Invocations du Programme* demeurent sur le serveur jusqu'à leur suppression par le *Client*. Lorsque l'Invocation d'un 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 d'un Programme* de ce type a exécuté un cycle ou a été redémarré depuis son demarrage (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 d'un Programme* de ce type peut exécuter un cycle ou être redémarrée depuis son demarrage (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 du Programme* de ce type qui existent réellement.

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

5.2.3 Composants type d'un programme

5.2.3.1 Vue d'ensemble

Les *Composants Type d'un Programme* comprennent un jeu de références aux instances d'objet de *StateTypes* (Types d'Etat), *TransitionTypes* (Types de Transition), *EventTypes* (Types d'Evénement) et les *Méthodes* qui dans leur ensemble définissent le *FiniteStateMachine* du *Programme*.

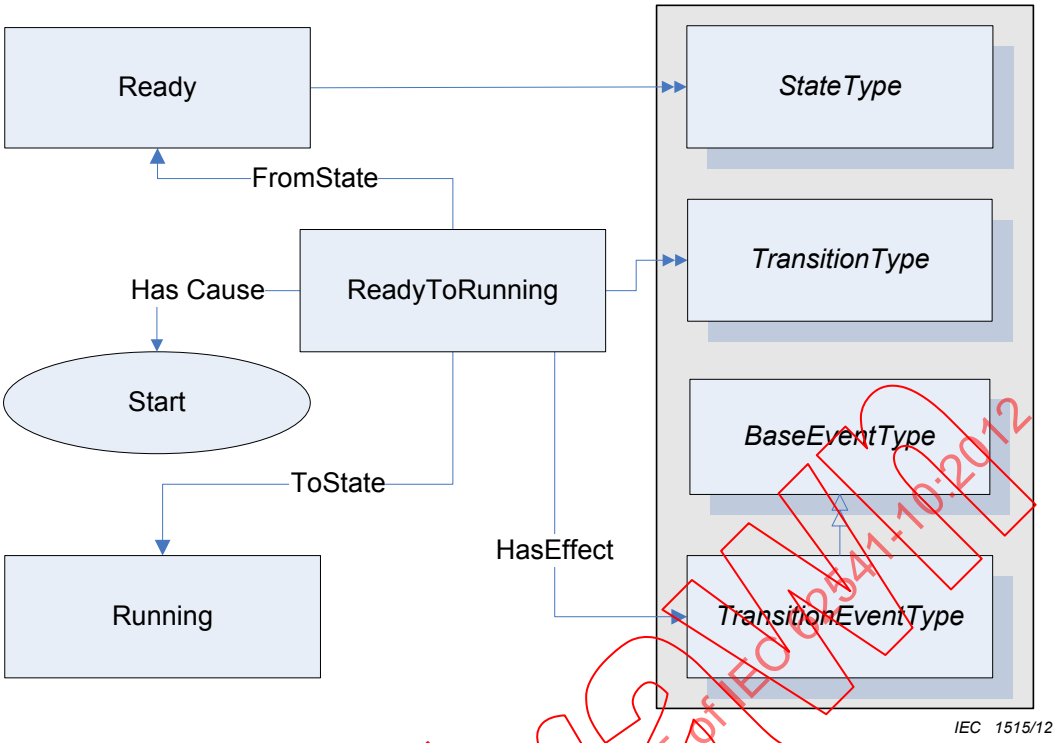


Figure 5 – Références FSM d'un programme

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 Référence) supplémentaires ont été omis pour simplifier la représentation.

5.2.3.2 États type d'un programme

Le Tableau 6 spécifie les *Objets* d'Etat *Type d'un Programme*. Ces *Objets* d'Etat sont les instances du *StateType* (Type d'Etat) définies dans la CEI 62541-5:2011, Annexe B. Une valeur unique *StateNumber* est attribuée à chaque Etat. 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 *FinitStateMachine*.

Tableau 6 – États d'un programme

Nom d'exploration	Références	Nom d'exploration cible	Valeur	Définition de type cible	NOTES
Etats					
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 d'un Programme ne 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*.

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 d'un programme

Les Transitions *Type d'un Programme* sont les instances du *TransitionType* (Type de Transition) définies dans la CEI 62541-5:2011 en 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 d'exploration	Références	Nom d'exploration 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			

Nom d'exploration	Références	Nom d'exploration cible	Valeur	Définition de type cible	NOTES
Transitions					
RunningToSuspended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	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.*

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 événements de type *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 la CEI 62541-7 nécessite le support du *AuditProgramTransitionEventType.*

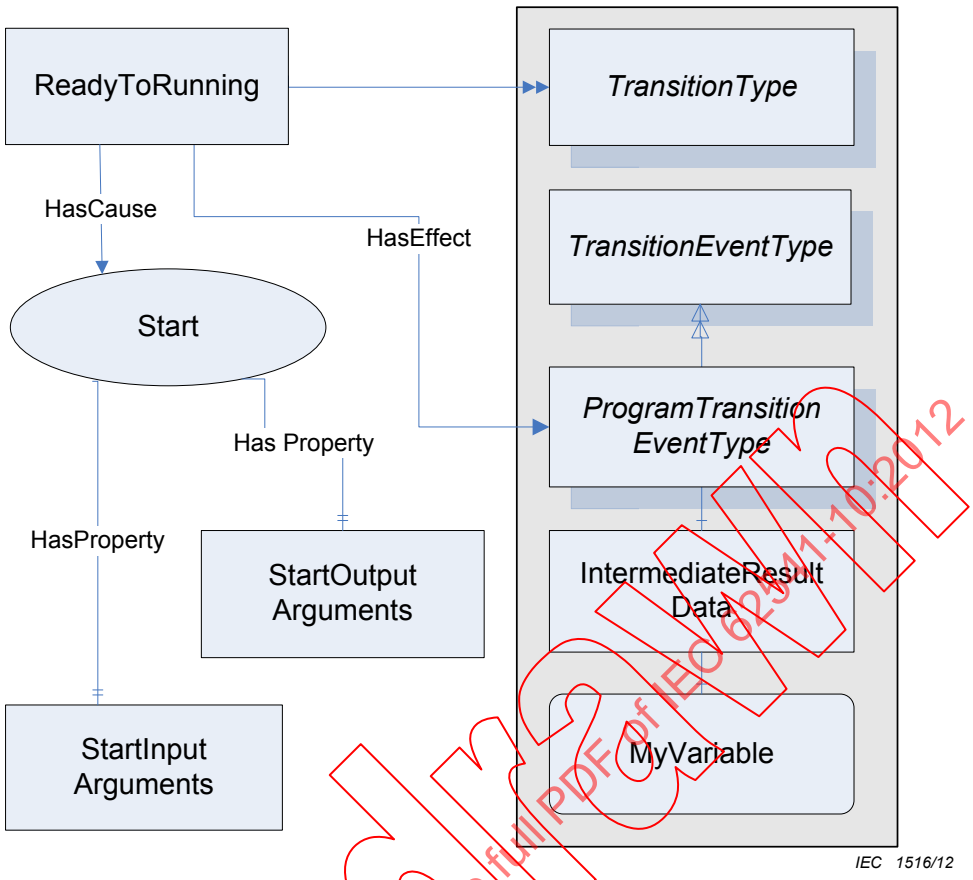


Figure 6 – Causes et effets type d'un programme

5.2.4 Causes type d'un Programme (Méthodes)

5.2.4.1 Vue d'ensemble

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

Tableau 8 – Causes type d'un programme

Nom d'exploration	Références	Nom d'exploration 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 du *CurrentState* (Etat 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 *CurrentState* 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 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 d'un 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 d'un Programme* de ce *Type* doivent satisfaire aux arguments spécifiés.

5.2.5 Effets type d'un programme (Evénements)

5.2.5.1 Vue d'ensemble

Le *Type d'un Programme* comprend des références de composant aux *Effets* de chacune des transitions d'états du *Programme*. Ces *Effets* sont des *Evénements.* Chaque *Transition* doit avoir une référence *HasEffect* à un *ProgramTransitionEventType* et peut avoir un *AuditProgramTransitionEventType.* Lorsque la transition se produit, des notifications d'événements du type référencé sont générées pour les *Clients* abonnés. L'Invocation d'un Programme peut être utilisée comme *EventNotifier* (Notificateur d'Evénements) de ces événements 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. Les *AuditProgramTransitionEventTypes* permettent d'auditer les modifications apportées à l'*Etat de Programme* conjointement aux *Appels de Méthode* du client.

5.2.5.2 ProgramTransitionEventType

Le *ProgramTransitionEventType* (Type d'Evénement 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 Programme).

Tableau 9 – ProgramTransitionEventType

Attribut	Valeur				
BrowseName	ProgramTransitionEventType				
IsAbstract	Vrai				
Références	**Classe de nœud**	**Nom d'exploration**	**Type de données**	**Définition de type**	**Règle de modélisation**
Hérite des *Propriétés* du *TransitionEventType* de base définies dans la CEI 62541-5:2011, B.4.16.					
HasComponent	Objet	IntermediateResult		BaseObjectType	Facultative

TransitionNumber identifie la *Transition d'un Programme* qui a déclenché *l'Evénement*.

FromStateNumber identifie l'*Etat* avant la *Transition d'un Programme*.

ToStateNumber identifie l'*Etat* 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. Le *ObjectType (Type d'Objet)* de *IntermediateResult (Résultat Intermédiaire)* spécifie l'ensemble des *Variables* utilisant un jeu de *Références HasComponent*.

Tableau 10 – ProgramTransitionEvents

Nom d'exploration	Références	Nom d'exploration cible	Va-leur	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	
	HasComponent	IntermediateResults		ObjectType	Facultatif
SuspendedToHaltedEvent					
	HasProperty	TransitionNumber	7	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	

Nom d'exploration	Références	Nom d'exploration cible	Va-leur	Définition de type cible	NOTES
Effets					
	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'Evénement 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'Etats du Programme* associées à toute *Méthode de Commande du Programme* invoquée par le *Client*. Les *Serveurs* doivent générer les *AuditProgramTransitionEvents* s'ils supportent le profil "Audit Server Facet" défini dans la CEI 62541-7.

Le Tableau 11 spécifie la définition d'*AuditProgramTransitionEventType*

Tableau 11 – AuditProgramTransitionEventType

Attribut	Valeur				
BrowseName	AuditProgramTransitionEventType				
IsAbstract	Vrai				
Références	**Classe de nœud**	**Nom d'exploration**	**Type de données**	**Définition de type**	**Règle de modélisation**
Hérite des *Propriétés* du *AuditUpdateStateEventType* définies dans la CEI 62541-5:2011, B.4.17.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *AuditUpdateStateEventType* défini dans la CEI 62541-5:2011, B.4.17 à l'exception de ce qui est noté ci-dessous CEI 62541-5.

La *Propriété Status (Etat),* spécifiée au 6.4.3 de la CEI 62541-5:2011 « *AuditEventType* » identifie si la transition d'états a pour cause un appel de Méthode de Commande de Programme (*Etat* mis à VRAI) ou non (*Etat* mis à FAUX).

SourceName (Nom Source) spécifié au 6.4.2 de la CEI 62541-5:2011 *BaseEventType* identifie la *Méthode* provoquant la *ProgramTransition (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 au 6.4.3 de la CEI 62541-5:2011 « *AuditEventType* », identifie l'utilisateur du *Client* ayant émis la *Méthode de Commande de Programme* si elle est associée à cette *Transition d'Etats de Programme*.

La *Propriété ActionTimeStamp,* spécifiée au 6.4.3 de la CEI 62541-5:2011 « *AuditEventType* », identifie le moment auquel se produit la *Transition d'Etats de Programme* ayant donné lieu à la génération de l'événement.

La *Propriété TransitionNumber* est une *Variable* qui identifie la Transition qui a déclenché l'événement.

5.2.7 FinalResultData

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

5.2.8 ProgramDiagnosticType

Le *ProgramDiagnoticType* (Type de Diagnostic d'un Programme) fournit des informations qui peuvent aider à diagnostiquer les problèmes liés au *Programme*. Cet objet comporte un ensemble de *Variables* qui détaille l'activité de *ProgramInvocation*. Le Tableau 12 spécifie les *Variables* qui constituent le *ProgramDiagnoticType*.

Tableau 12 – ProgramDiagnosticType

Attribut	Valeur			
BrowseName	ProgramDiagnosticsType			
IsAbstract	Faux			
Références	**Classe de nœud**	**Nom d'exploration**	**Type de données / Définition de type**	**Règle de modélisation**
Sous-type du BaseObjectType défini dans la CEI 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	Int32	Obligatoire
HasComponent	Variable	LastMethodInputArguments	InputArguments	Obligatoire
HasComponent	Variable	LastMethodOutputArguments	OutputArguments	Obligatoire
HasComponent	Variable	LastMethodCallTime	UTCTime	Obligatoire
HasComponent	Variable	LastMethodReturnStatus	returnStatus	Obligatoire

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 d'un Programme.*

CreateClientName est le nom du client de la session ayant créé l'*Invocation d'un Programme*.

InvocationCreationTime identifie le moment où l'*Invocation d'un Programme* a été créée.

LastTransitionTime identifie le moment où la dernière transition d'états de programme a eu lieu.

LastMethodCall identifie la dernière *Méthode de Programme* appelée sur l'*Invocation d'un Programme*.

LastMethodSessionId comporte le *SessionId* de la session au cours de laquelle le dernier appel de Méthode de Commande de Programme pour l'*Invocation d'un Programme* a été émis.

LastMethodClientName est le nom du client de la session ayant effectué le dernier appel de Méthode pour l'*Invocation d'un Programme*.

LastMethodInputArguments conservent les valeurs des arguments d'entrée sur le dernier appel de *Méthode du Programme*.

LastMethodOutputArguments conservent les valeurs des arguments de sortie sur le dernier appel de *Méthode du Programme*.

LastMethodCallTime identifie le moment du dernier appel de Méthode pour l'*Invocation d'un Programme*.

LastMethodReturnStatus conserve la valeur du *returnStatus* pour la dernière *Méthode de Commande de Programme* demandée pour cette *Invocation du Programme*.

Annexe A
(informative)

Exemple de Programme

A.1 Vue d'ensemble

Cet exemple illustre l'utilisation d'un *Programme* OPC UA 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.

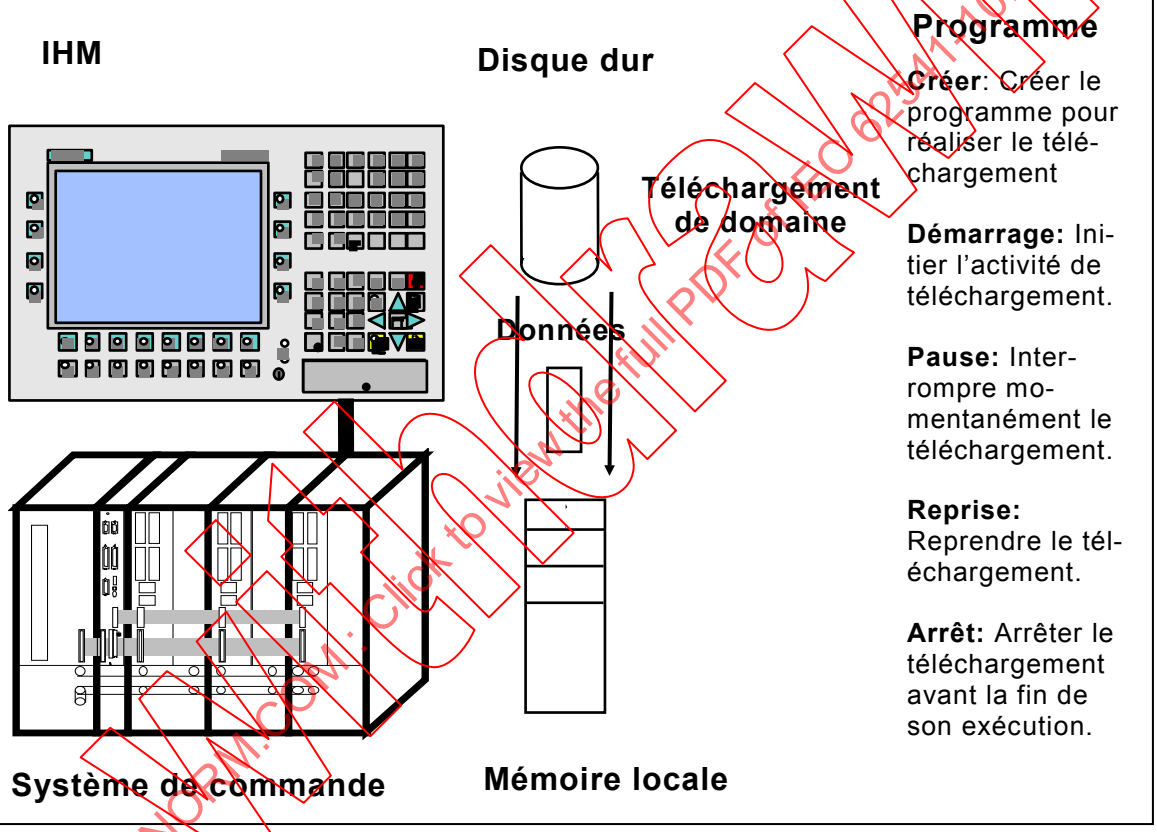


Figure A.1 – Exemple de programme

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* OPC *UA*.

A.2 Programme DomainDownload

A.2.1 Généralités

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