OPC Unified Architecture

Specification

Part 10: Programs

Version 1.00

January 29, 2007

CONTENTS

1 Scope

This specification 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 *NodeClass*es and *Attributes* is specified in [UA Part 3]. The services such as those used to invoke the *Methods* used to manage P*rograms* are specified in [UA Part 4].

2 Reference documents

[UA Part 1] OPC UA Specification: Part 1 – Concepts, Version 1.0 or later
http://www.opcfoundation.org/UA/Part1/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services, Version 1.0 or later
http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model, Version 1.1 or later
http://www.opcfoundation.org/UA/Part5/

[UA Part 7] OPC UA Specification: Part 7 – Profiles, Version 1.0 or later
http://www.opcfoundation.org/UA/Part7/

3 Terms, definitions, and abbreviations

3.1 OPC UA Part 1 terms

The following terms defined in [UA Part 1] of this multi-part specification apply.

1) EventType
2) Information Model
3) Method
4) Node
5) NodeClass
6) Notification
7) Object
8) ObjectInstance
9) ObjectType
10) Program
11) Reference
12) ReferenceType
13) Service
14) Session
15) Subscription

16) Variable

3.2 OPC UA Part 3 terms

The following terms defined in [UA Part 3] apply.

1. TypeDefinition Node
2. Property
3. Variable

3.3 OPC UA Program terms

The following terms are defined for UA Programs.

3.3.1 Function

A Function is a programmatic task performed at a server or device, usually accomplished by computer code execution.

3.3.2 Program

A Program is a complex *Function* in a server or underlying system that can be invoked and managed by an OPC *UA Client*.

3.3.3 Finite State Machine

A Finite State Machine is a 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.

3.3.4 ProgramType

A ProgramType is an *ObjectType Node* that represents the type definition of a *Program* and is a subtype of the *StateMachineType*.

3.3.5 Program Control Method

A Program Control Method *is a Method* specified by this specification having specific semantics designed for the control of a *Program* by causing a state transition.

3.3.6 Program Invocation

A Program Invocation is a unique *Object* instance of a *Program* exiting on a *Server.* The Program Invocation is distinguished from other *Object* instances of the same *ProgramType* by the object node's unique browse path.

3.4 Abbreviations and symbols

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. 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.

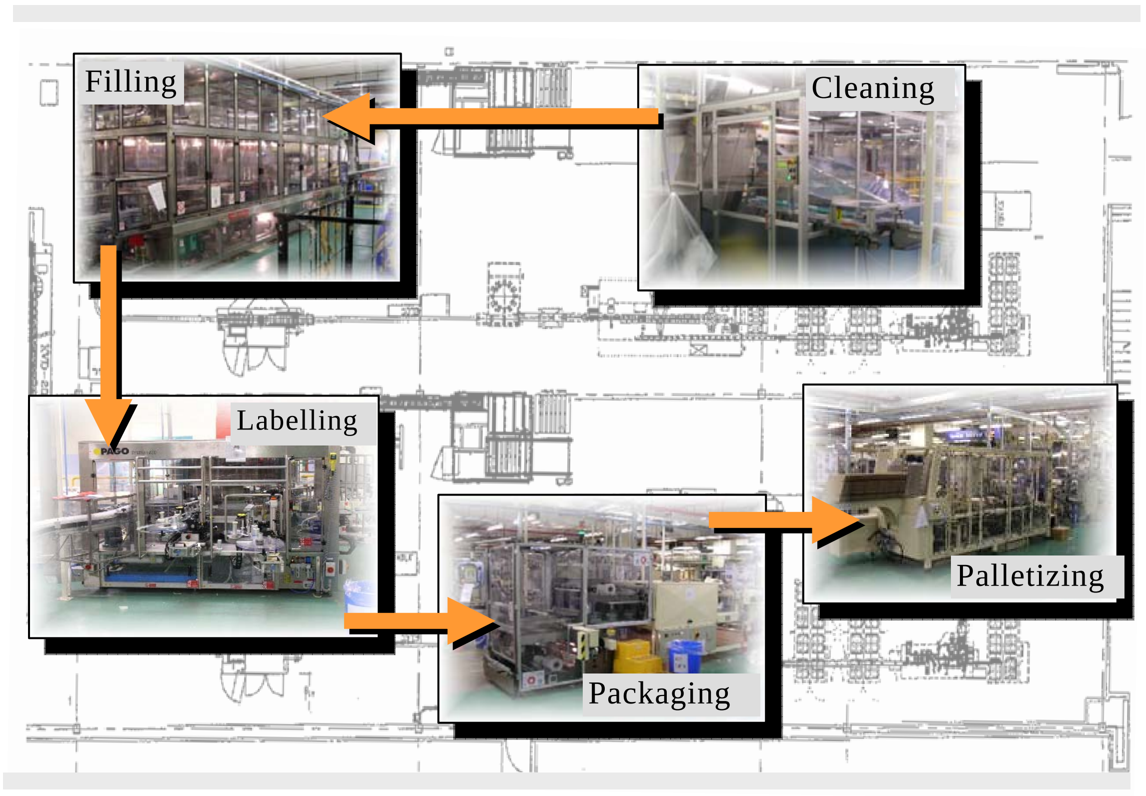

Figure 1 - Automation Facility Control

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 [UA Part 3] – Address Space Model and [UA Part 4] - Services. This specification describes the extensions to, or specific use of the core capabilities defined in the first seven parts of the OPC UA multi-part specification required for *Programs*. Support for the feature described in this specification are described in [UA Part 7] - Profiles

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.

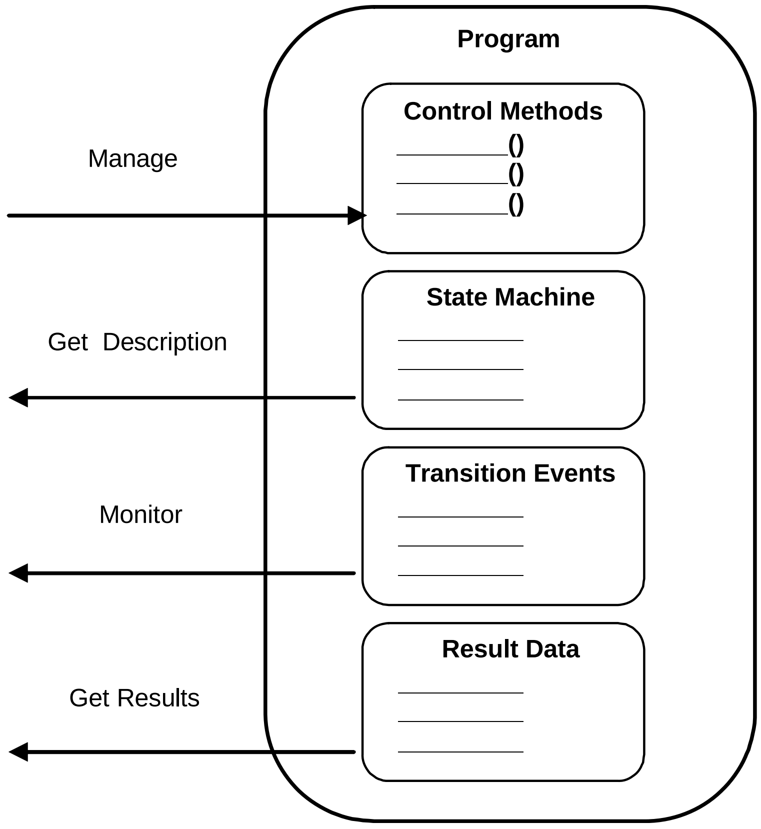

Figure 2 - Program Illustration

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 statues, 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

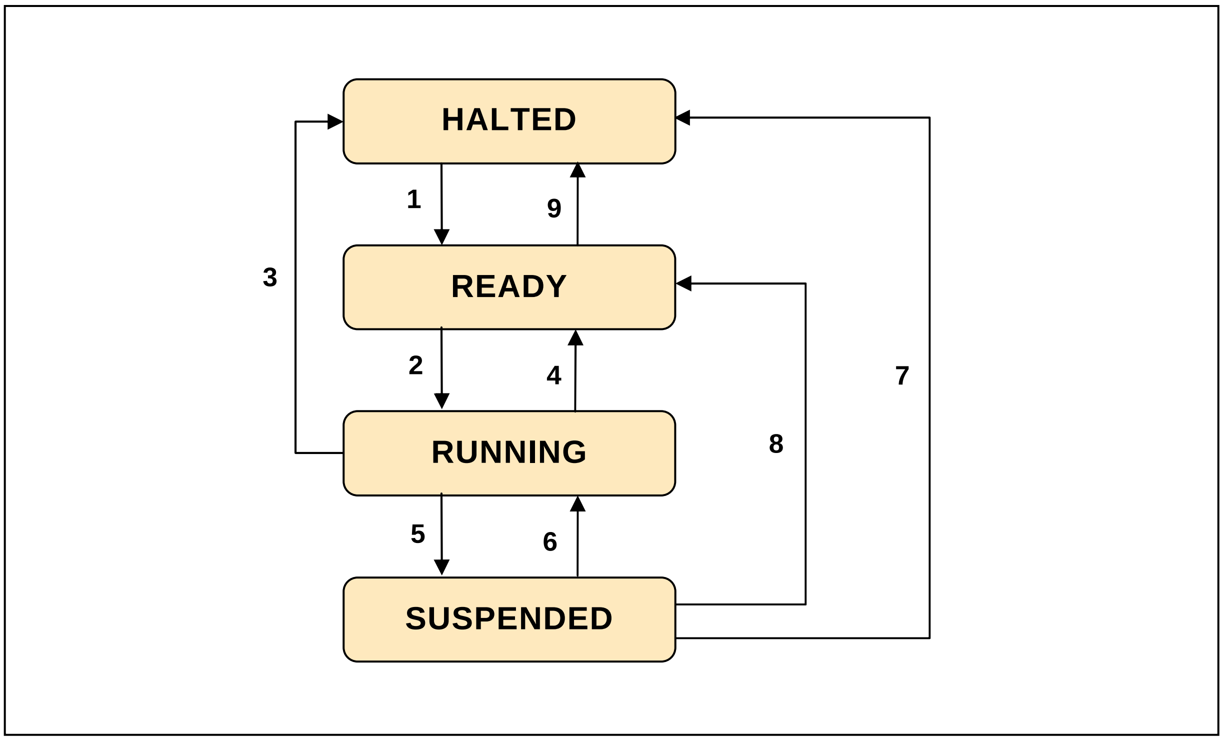

Figure 3 - Program States and Transitions

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* must support this base set. A *Program* may or may not require a client action to cause the state to change.

Table 2 - Program States

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

The set of states defined to describe a *Program* can be expanded. Program sub states can be defined for the base states to provide more resolution 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.

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 must 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* must 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 [UA Part 4] 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 [UA Part 3] 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 *Object DeleteNode Service* defined in [UA Part 4] is used to remove the *Program* Instance. The *Program* must 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 *StateMachineType and* basic *ObjectType* Models presented in [UA Part 5]. Each *Program* has a type definition that is the subtype of the *StateMachineType.* 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*.

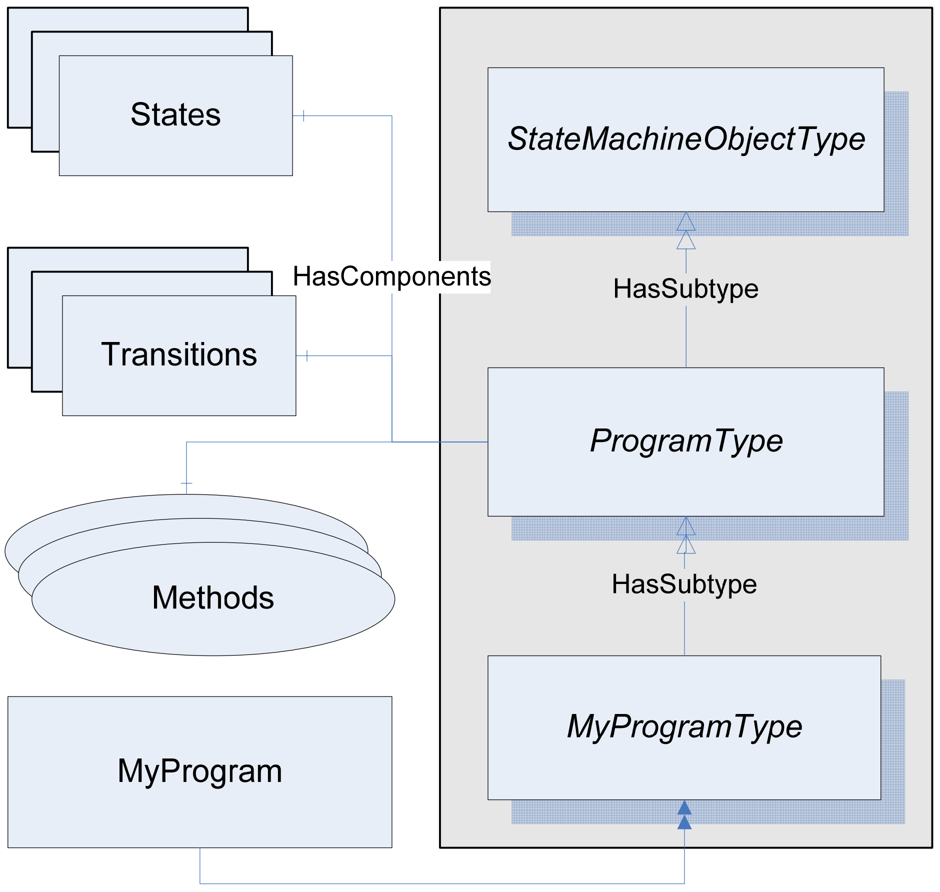

Figure 4 - Program Type

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

5.2 ProgramType

5.2.1 Overview

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

Table 5 - ProgramType

Attribute	**Value**				
	Includes all attributes specified for the StateMachineType				
BrowseName	ProgramType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	Creatable	Boolean	PropertyType	None
HasProperty	Variable	Deletabe	Boolean	PropertyType	New
HasProperty	Variable	AutoDelete	Boolean	PropertyType	Shared
HasProperty	Variable	RecycleCount	Int32	PropertyType	New
HasProperty	Variable	InstanceCount	UInt32	PropertyType	None
HasProperty	Variable	MaxInstanceCount	UInt32	PropertyType	None
HasProperty	Variable	MaxRecycleCount	UInt32	PropertyType	None
HasComponent	Variable	ProgramDiagnostic		ProgramDiagnosticType	OptionalNew
InitialState	Object	(Halted or Ready)		StateType	
HasComponent	Object	Halted		StateType	New
HasComponent	Object	Ready		StateType	New
HasComponent	Object	Running		StateType	New
HasComponent	Object	Suspended		StateType	New
HasComponent	Object	HaltedToReady		TransitionType	New
HasComponent	Object	ReadyToRunning		TransitionType	New
HasComponent	Object	RunningToHalted		TransitionType	New
HasComponent	Object	RunningToReady		TransitionType	New
HasComponent	Object	RunningToSuspended		TransitionType	New
HasComponent	Object	SuspendedToRunning		TransitionType	New
HasComponent	Object	SuspendedToHalted		TransitionType	New
HasComponent	Object	SuspendedToReady		TransitionType	New
HasComponent	Object	ReadyToHalted		TransitionType	New
HasComponent	Method	Start			OptionalNew
HasComponent	Method	Suspend			OptionalNew
HasComponent	Method	Reset			OptionalNew
HasComponent	Method	Halt			OptionalNew
HasComponent	Method	Resume			OptionalNew
HasComponent	Object	FinalResultData		BaseObjectType	OptionalNew

5.2.2 ProgramType Properties

The *Creatable Property* is a Boolean that specifies if *Program Invocations* of this *ProgramType* can be created by an OPC 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 an OPC 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.*

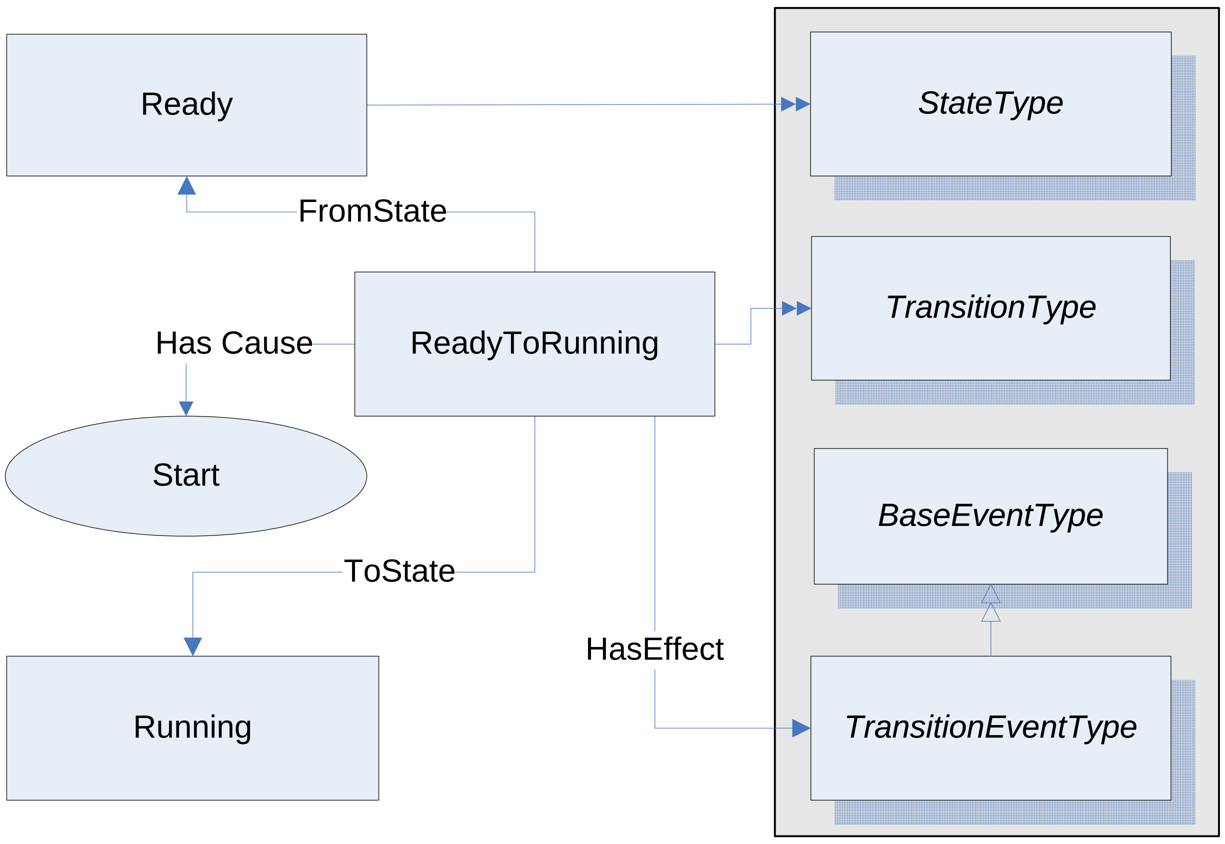

Figure 5 - Program FSM References

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 [UA Part 5] - SM Appendix. 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 Ready 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* state and from the *Ready*, *Running*, and *Suspended* States.

5.2.3.2.1 Program Initial State

The initial state of a *ProgramType* is the state that the Program Invocation assumes upon creation. This *State* is formally specified in the *Program Finite State Machine* by defining an *InitialState Reference* in the *ProgramType*. The *InitialState Reference Type* is defined in [UA Part 5] - SM Appendix. If specified, the target of the *InitialState Reference* must be either the *Halted State* or the *Ready State.* In some *Programs*, the initial state can vary and would not be referenced. If these *Halted* or *Ready States* contain subordinate *Finite State Machines*, the FSM can include an InitalState reference.

5.2.3.3 ProgramType Transitions

ProgramType Transitions are instances of *Objects* of the *TransitionType* defined in [UA Part 5] - SM Appendix 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*.

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	ProgramTransitionAuditEventType			Optional
ReadyToRunning	HasProperty	TransitionNumber	2	PropertyType	
	ToState	Running		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
RunningToHalted	HasProperty	TransitionNumber	3	PropertyType	
	ToState	Halted		StateType	
	FromState	Running		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
RunningToReady	HasProperty	TransitionNumber	4	PropertyType	
	ToState	Ready		StateType	
	FromState	Runnning		StateType	
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
RunningToSuspended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
SuspendedToRunning	HasProperty	TransitionNumber	6	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
SuspendedToHalted	HasProperty	TransitionNumber	7	PropertyType	
	ToState	Halted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
SuspendedToReady	HasProperty	TransitionNumber	8	PropertyType	
	ToState	Ready		StateType	
	FromState	Suspended		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
ReadyToHalted	HasProperty	TransitionNumber	9	PropertyType	
	ToState	Halted		StateType	
	FromState	Ready		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional

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.* When this transition occurs,

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 by the *Halt Method.*

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 *ProgramTransitionAuditEventType* defined in xxx*.* The *ProgramTransitionEventType* notifies *Clients* of the Program Transition and conveys result data. The *ProgramTransitionAuditEventType* can optionally be used to audit transitions that result from *Program Control Methods*.

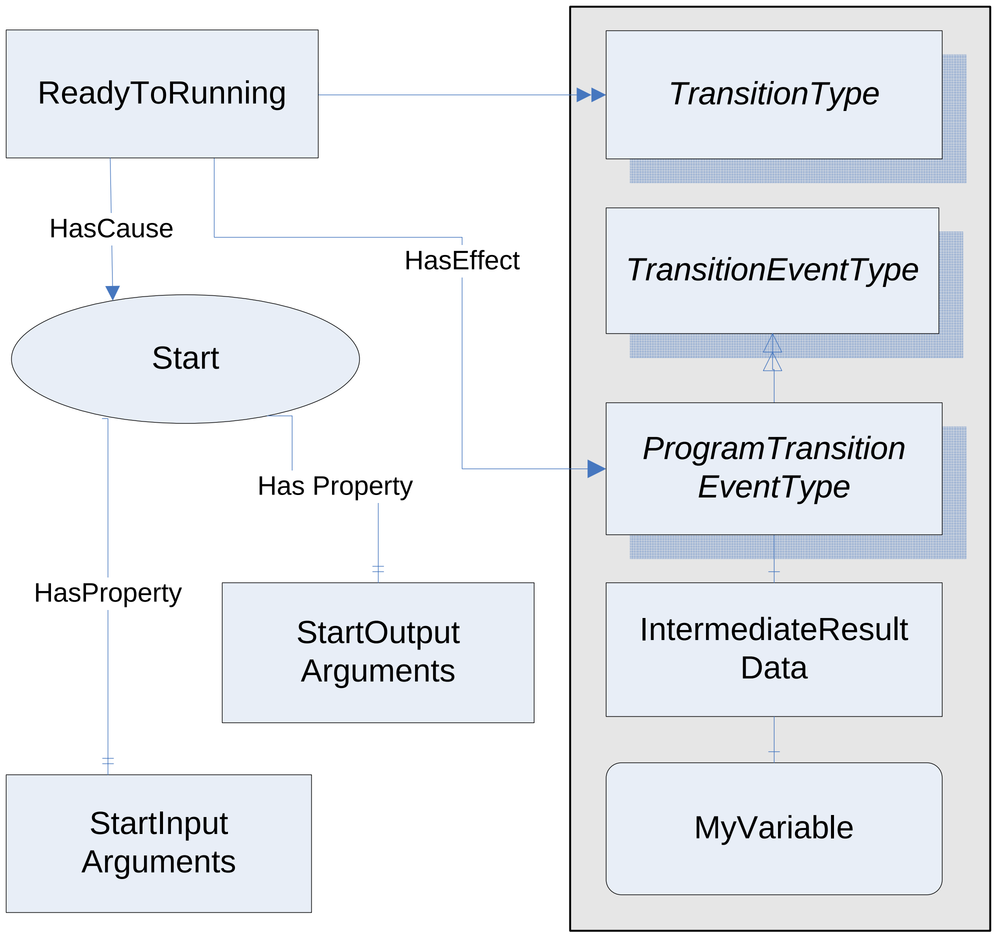

Figure 6 - ProgramType Causes and Effects

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 - ProgramType Causes specifies the *Methods defined as Causes for ProgramTypes.* Figure 6 - ProgramType Causes and Effects 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
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* must 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* must 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* must have a *HasEffect* reference to a

ProgramTransitionEventType and can have a ProgramTransitionAuditEventType. 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. ProgramTransitionAuditEventTypes* allows the auditing of changes to the *Program*'s *State* in conjunction with auditing 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**	**ModellingRule**
Inherits the *Properties* of the base *TransitionEventType* defined in [UA- Part 5] SM appendix.					
HasComponent	Object	IntermediateResult		BaseObjectType	OptionalNew

The *TransitionNumber* property is a *Variable* that identifies the *Program Transition* that triggered the event.

The FromStateNumber property is a *Variable* that is the *StateNumber* of the originating state of the *Program Transition*.

The ToStateNumber property is a *Variable* that is the *StateNumber* of the terminal state in 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	
	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 ProgramTransitionAuditEventType

The *ProgramTransitionAuditEventType* is a subtype of the *AuditEventType*. This *EventType* inherits all *Properties* of the *AuditEventType* defined in [UA- Part 5]. It is used with *Programs* to provide a means to audit the *Program State Transitions* associated with any *Client* invoked *Program Control Method*. Table 11 specifies the definition of the *ProgramTransitionAuditEventType*

Table 11 - ProgramTransitionAuditEvent

Attribute	**Value**				
BrowseName	ProgramTransitionAuditEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherits the *Properties* of the *AuditEventType* defined in [UA- Part 5]					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	New

The *Status Property,* specified in [UA- Part 5] *AuditEventType*, identifies whether the state transition resulted from a Program Control Method call (set *Status* to TRUE) or not (set *Status* to FALSE).

The *ClientUserId Property,* specified in [UA- Part 5] *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 [UA- Part 5] *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. The values of the Variables

5.2.8 ProgramDiagnosticType

5.2.8.1 Overview

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 [UA Part 5].				
HasComponent	Variable	CreateSessionId	Int32	New
HasComponent	Variable	CreateClientName	String	New
HasComponent	Variable	InvocationCreationTime	UTCTime	New
HasComponent	Variable	LastTransitionTime	UTCTime	New
HasComponent	Variable	LastMethodCall	String	New
HasComponent	Variable	LastMethodSessionId	Int32	New
HasComponent	Variable	LastMethodInputArguments	InputArguments	New
HasComponent	Variable	LastMethodOutputArguments	OutputArguments	New
HasComponent	Variable	LastMethodCallTime	UTCTime	New
HasComponent	Variable	LastMethodReturnStatus	returnStatus	New

The *CreateSessionId* contains the *SessionId* of the session on which the call to the *Create Service* 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 - Program Example

A.1 Overview

This example illustrates the use of a UA *Program* to manage a domain download into a control system as depicted in Figure ***7***. The download requires the segmented transfer of control operation data from a secondary storage device to the local memory within a control system.

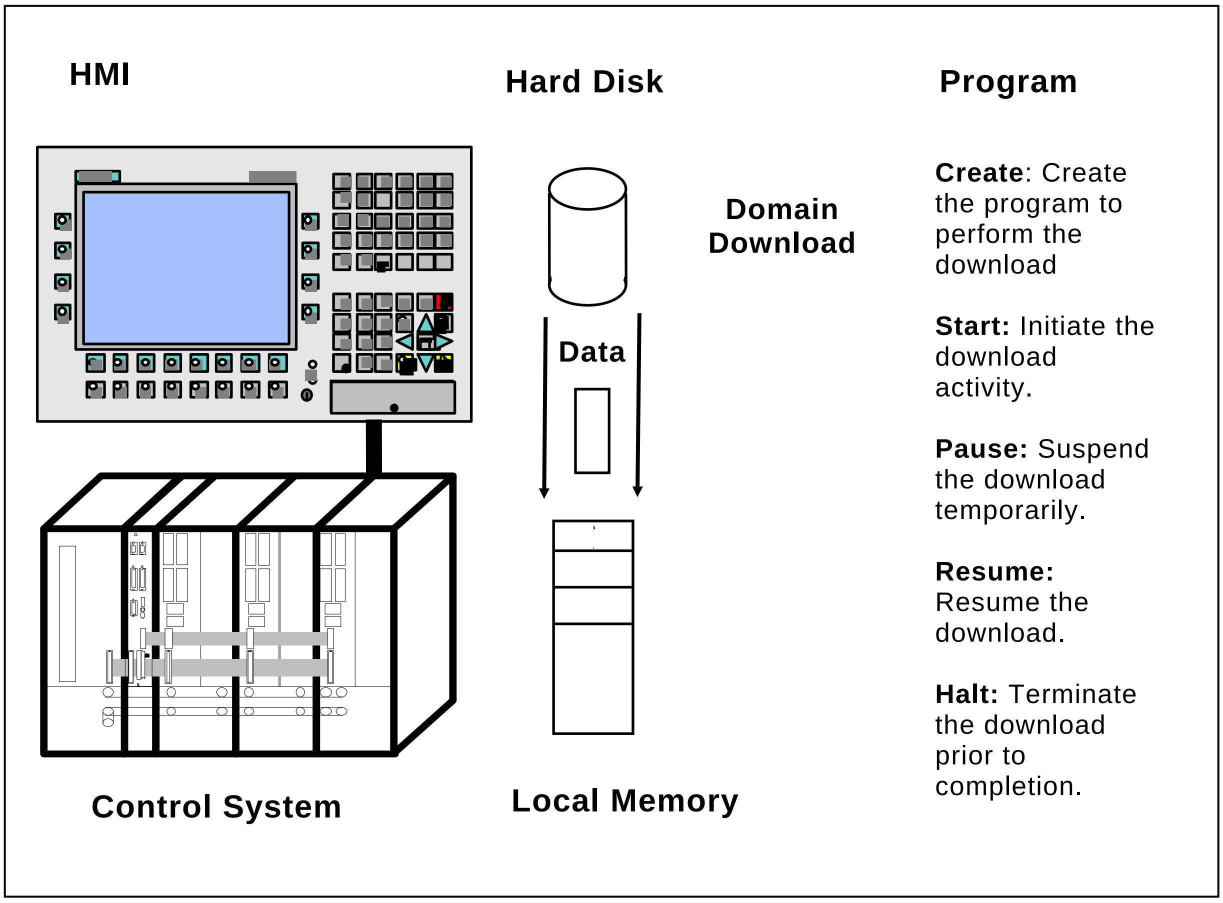
HMI
Hard Disk
Program
Domain Download
Data
Control System
Local Memory
Create: Create the program to perform the download
Start: Initiate the download activity.
Pause: Suspend the download temporarily.
Resume: Resume the download.
Halt: Terminate the download prior to completion.

Figure *7* - Program Example 1

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 *UA Server*.

A.2 DomainDownload Program

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

A.2.1 DomainDownload States

The basic state model for the DomainDownload *Program* is presented in ***Figure 8***. 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 *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 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.

Error! Objects cannot be created from editing field codes.

Figure 8 – DomainDownload State Diagram

Figure 8 illustrates the sequence of state transitions. 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 13 along with t he state transitions.

A.2.2 DomainDownload Transitions

The valid state transitions specified for the DomainDownload *Program* and are specified in Table 13. 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, Ready to Running and to Opening occurs at the same time.

The table also specifies the defined states, causes for the transitions, and the effects of each transition. Program Control Methods are used by the *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 13 - 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	SuspandedToSending	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.2.3 DomainDownload Methods

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.2.3.1 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.2.4 DomainDownload Events

A *ProgramTransitionEventType* is specified for each of the DomainDownload *Program* transitions. The event types trigger a specific event notification to the *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.2.4.1 Event Information

The SendingToSending program transition event relays intermediate result data to the *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 *UA Client*.

A.2.4.2 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.2.5 DomainDownload Model

A.2.5.1 Overview

The 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 *UA* components. These components can be browsed by a *UA Client* to interpret or validate the actions of the Program.

A.2.5.2 DomainDownloadType

The DomainDownloadType is a subtype derived from the *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 14 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.

Figure 9 illustrates the components of DomainDownloadType and its instance, including their ownership and derivation. Transitions and EventTypes are represented by their numeric Identifiers in the figure. The references that exist between *StatesTypes*, *TransitionTypes*, *EventNotificationTypes*, and *Methods* are not shown in Figure 9.

Table 14 – 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**
HasSubStateMachine	Object	TransferStateMachine		StateMachineType	New
HasSubStateMachine	Object	FinishStateMachine		StateMachineType	New
HasComponent	Variable	ProgramDiagnostic		ProgramDiagnosticType	New
InitialState	Object	Ready		StateType	
HasComponent	Object	ReadyToRunning		TransitionType	New
HasComponent	Object	RunningToHalted		TransitionType	New
HasComponent	Object	RunningToSuspended		TransitionType	New
HasComponent	Object	SuspendedToRunning		TransitionType	New
HasComponent	Object	SuspendedToHalted		TransitionType	New
HasComponent	Method	Start			New
HasComponent	Method	Suspend			New
HasComponent	Method	Halt			New
HasComponent	Method	Resume			New
HasComponent	Object	FinalResultData		BaseObjectType	New

Table 15 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 15 - Transfer State Machine Type

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

Figure 15 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 16 - 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 17 specifies the Finish State Machine type that is a sub state machine of the DomianDownload *ProgramType*. This State Machine Type definition identifies the State types that compose the sub states for the Program's Halted State type.

Table 17 - Finish State Machine Type

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

Table 18 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 18 - 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 19 specifies constraining behaviour of a DomainDownload.

Table 19 – DomainDownload Type Property Attributes Variable Values

NodeClass	**BrowseName**	**Data Type**	**Data Value**	**Modelling Rule**
Variable	Creatable	Boolean	True	None
Variable	Deletabe	Boolean	True	New
Variable	AutoDelete	Boolean	False	Shared
Variable	RecycleCount	Int32	0	New
Variable	InstanceCount	UInt32	PropertyType	None
Variable	MaxInstanceCount	UInt32	500	None
Variable	MaxRecycleCount	UInt32	0	None

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

Figure **9** 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.

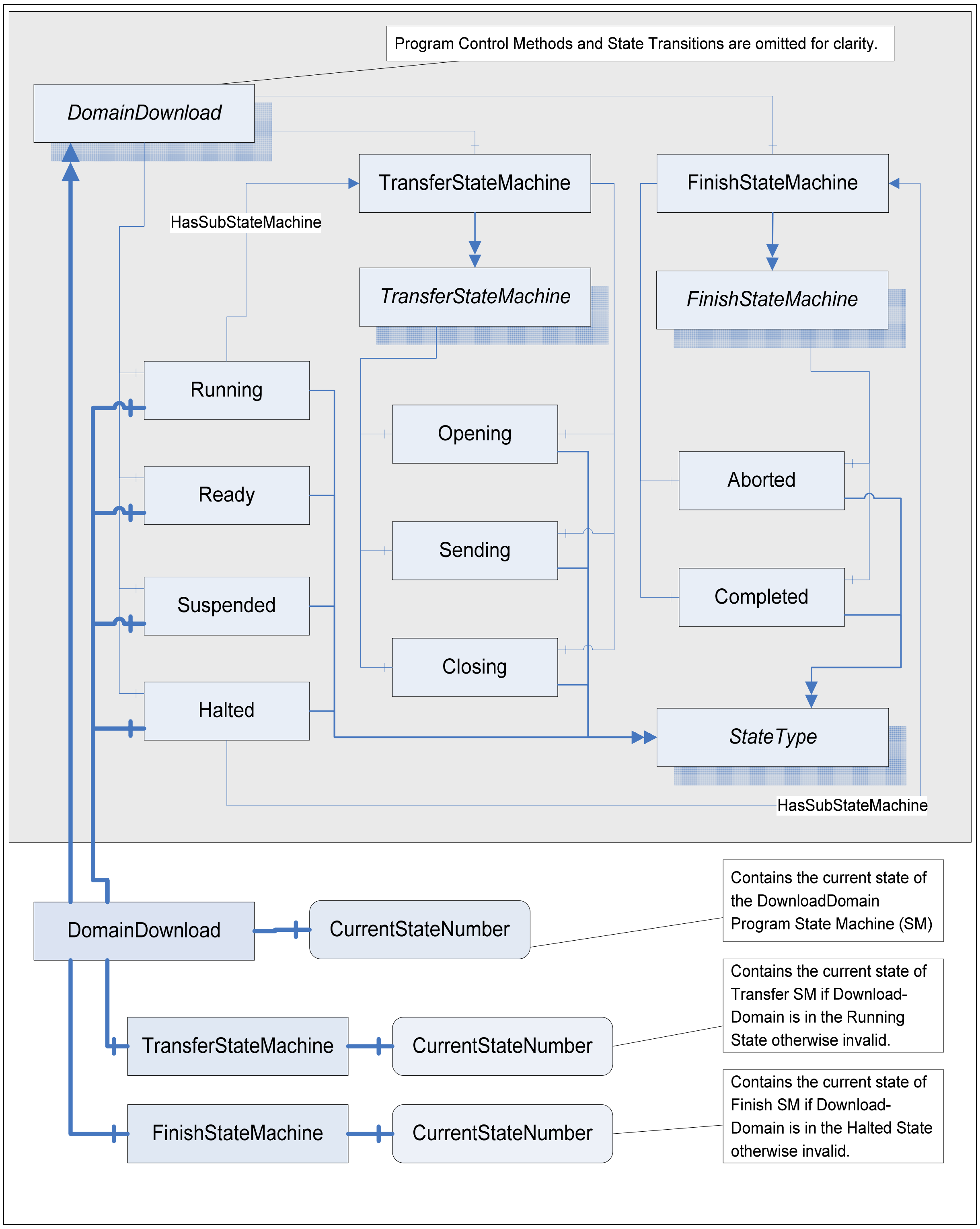
Program Control Methods and State Transitions are omitted for clarity.
DomainDownload
TransferStateMachine
FinishStateMachine
HasSubStateMachine
TransferStateMachine
FinishStateMachine
Running
Ready
Suspended
Halted
Opening
Sending
Closing
Aborted
Completed
StateType
HasSubStateMachine
DomainDownload
CurrentStateNumber
Contains the current state of the DownloadDomain Program State Machine (SM)
TransferStateMachine
CurrentStateNumber
Contains the current state of Transfer SM if Download-Domain is in the Running State otherwise invalid.
FinishStateMachine
CurrentStateNumber
Contains the current state of Finish SM if Download-Domain is in the Halted State otherwise invalid.

Figure 9 - DomainDownloadType Partial State Model

Table 20 specifies the *ProgramTransitionTypes* that are defined in addition to the Standard *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 20 - 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			
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	Closing		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
ClosingToCompleted	HasProperty	TransitionNumber	14	PropertyType	
	ToState	Completed		StateType	
	FromState	Closing		StateType	
	HasEffect	ProgramTransitionEventType			
SendingToSuspended	HasProperty	TransitionNumber	15	PropertyType	
	ToState	Suspended		StateType	
	FromState	Sending		StateType	
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
SuspendedToSending	HasProperty	TransitionNumber	16	PropertyType	
	ToState	Sending		StateType	
	FromState	Suspended		StateType	
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
SuspendedToAborted	HasProperty	TransitionNumber	18	PropertyType	
	ToState	Aborted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional
ReadyToOpening	HasProperty	TransitionNumber	17	PropertyType	
	ToState	Opening		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	ProgramTransitionAuditEventType			Optional

Figure 10 Through ***Figure 16*** illustrate portions of the DomainDownloadType model. In each figure, the referenced States, Methods, Transitions and EventTypes are identified for one or two state transitions.

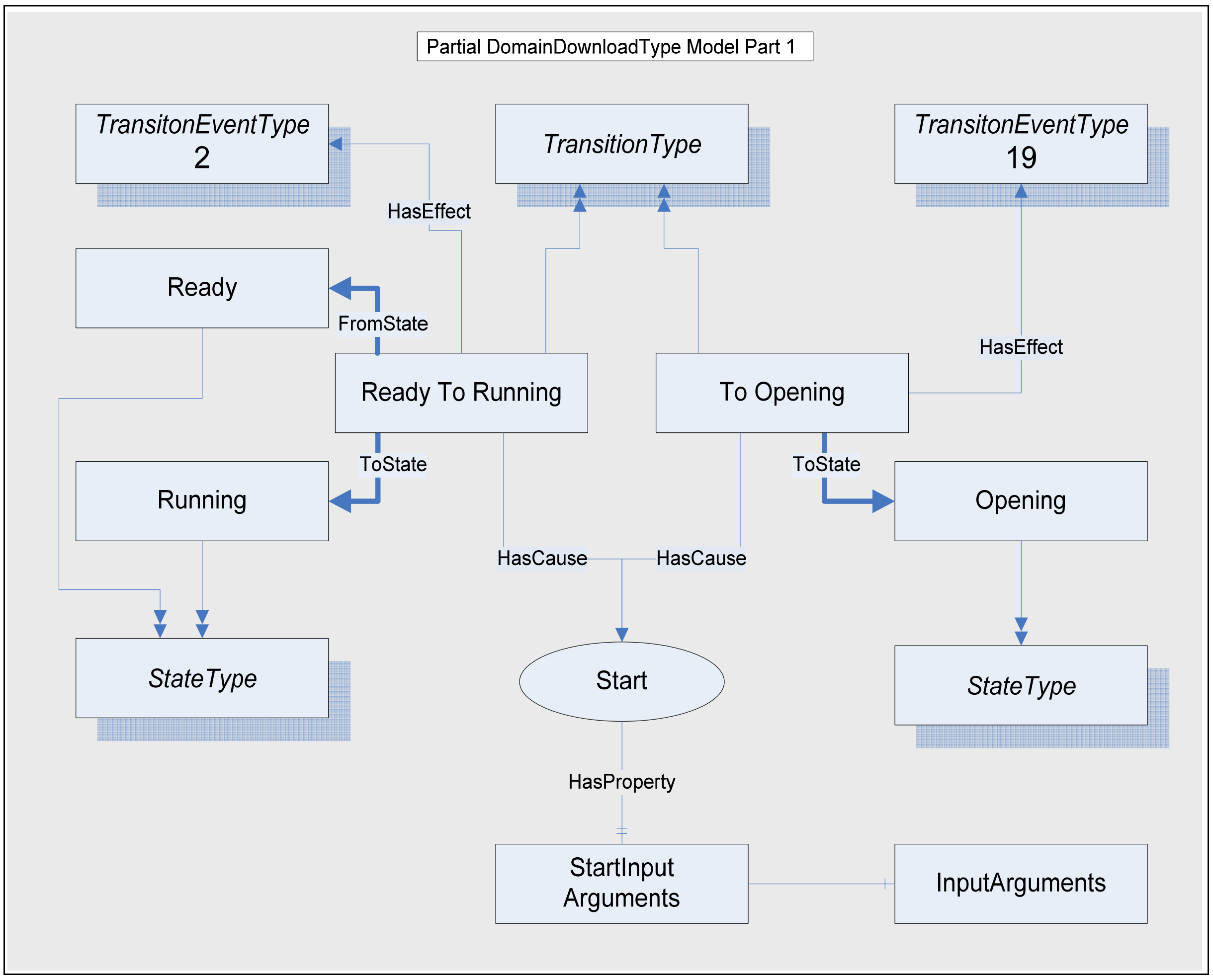

Figure 10 – Ready To Running Model

Figure 10 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 *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 *UA Server* issues two event notifications, ReadyToRunning (2) and ToOpening (19).

Table 21 - Start Method Additions

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

Table 21 specifies that the Start Method for the DomainDownloadType requires input arguments. Table 22 identifies the Start Arguments required.

Table 22 - StartArguments

Name	Type	Value
Argument 1	structure	
name	String	SourcePath
dataType	NodeId	StringNodeId
arraySize	Int32	-1
description	LocalizedText	The source specifier for the domain.
Argument 2	structure	
Name	String	DesinationPath
dataType	NodeId	StringNodeId
arraySize	Int32	-1
description	LocalizedText	The destination specifier for the domain.
Argument 3	structure	
name	String	DomainName
dataType	NodeId	StringNodeId
arraySize	Int32	-1
description	LocalizedText	The name of the domain.

Figure 11 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.

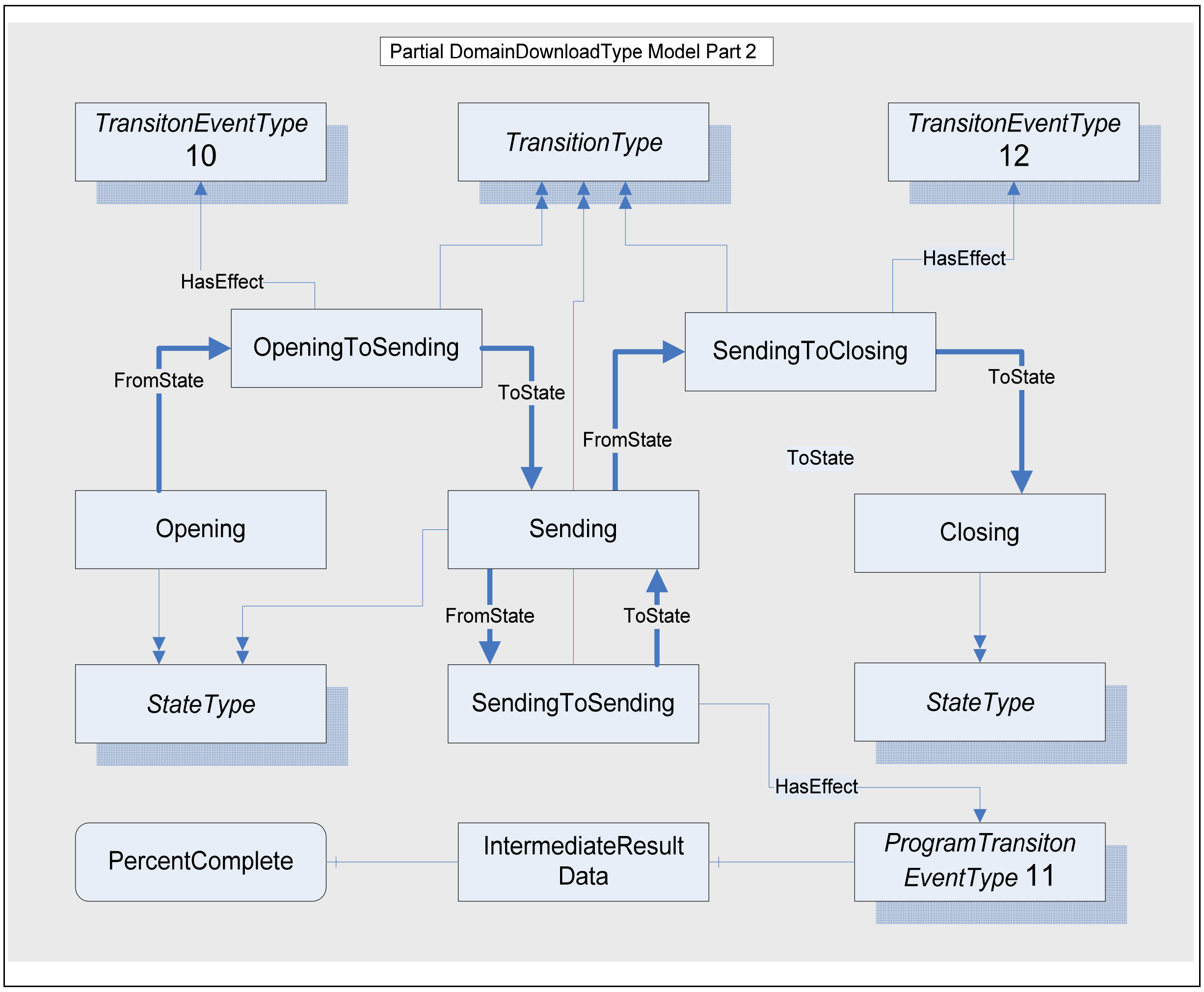

Figure 11 - Opening To Sending To Closing Model

Notice that a state transition can initiate and terminate at the same state (Sending). In this case the transition serves a purpose. The ProgramTransitionEventType effect referenced by the SendingToSending State Transition has an IntermediateResultData object reference. The

IntermediateResultData object serves to identify two variables whose values are obtained each time the state transition occurs. The values are sent to the *UA Client* with the event notification.

Table 24 identifies these variables as AmountTransferred and PercentageTransferred.

Table 23 - 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	New
HasComponent	Variable	PercentageTransferred	Long	VariableType	New

Table 24 - Immediate Result Data Variables

Immediate 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..

Figure 12 Illustrates the model for the Running To Suspended state transition. The cause for this transition is the *Suspend Method.* The *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.

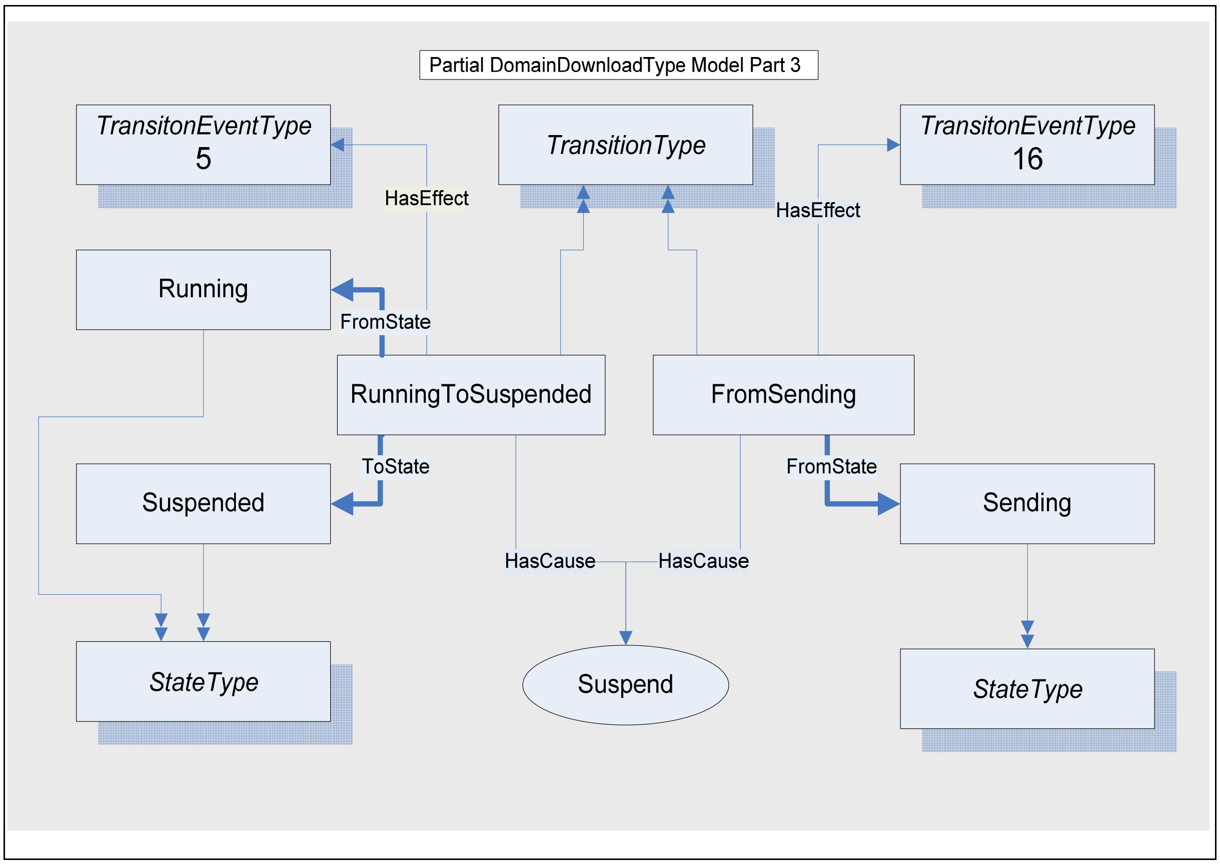

Figure 12 - Running To Suspended Model

Figure 13 illustrates the model for the Suspended To Running state transition. The cause for this transition is the *Resume Method.* The *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. Now that the Running state is active, the Sending State of the Transfer State Machine is again specified for the CurrentStateNumber.

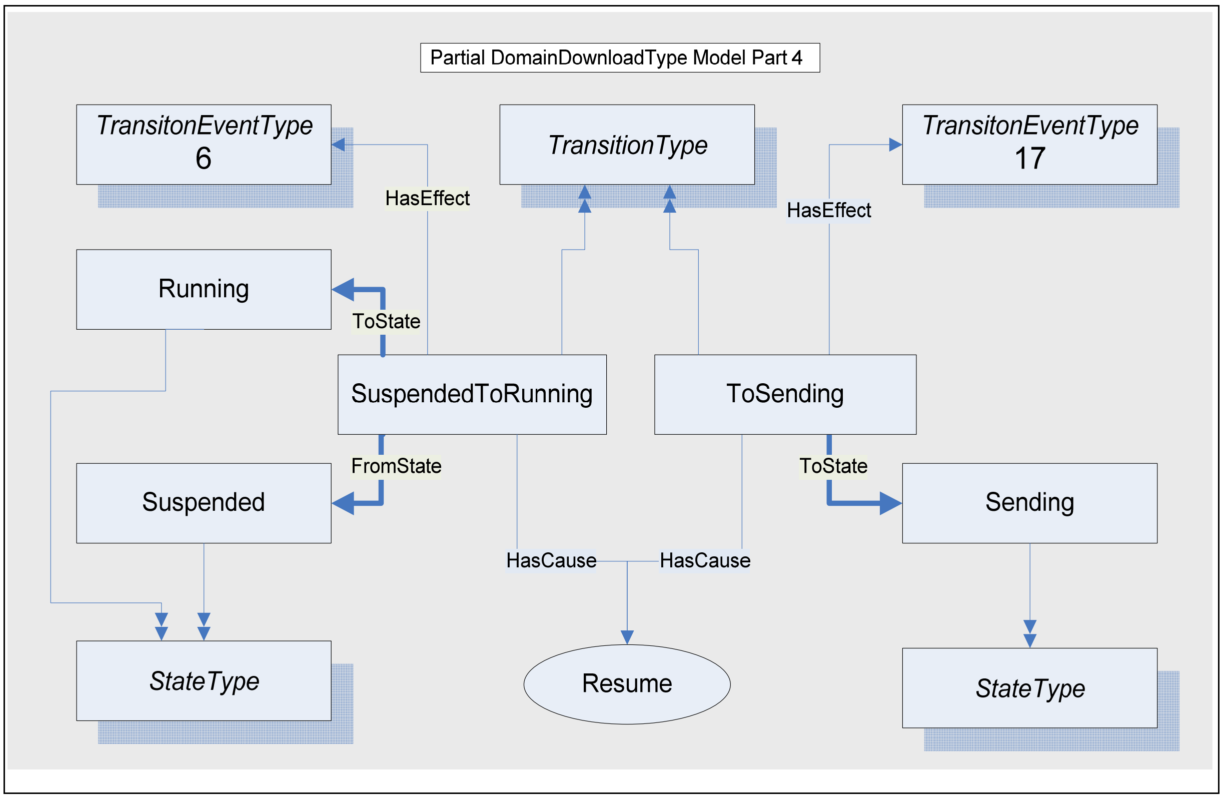

Figure 13 - Suspended To Running Model

Figure 14 illustrates the model for the Running To Halted state transition for an abnormal termination of the domain download. The cause for this transition is the *Halt Method.* The *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.

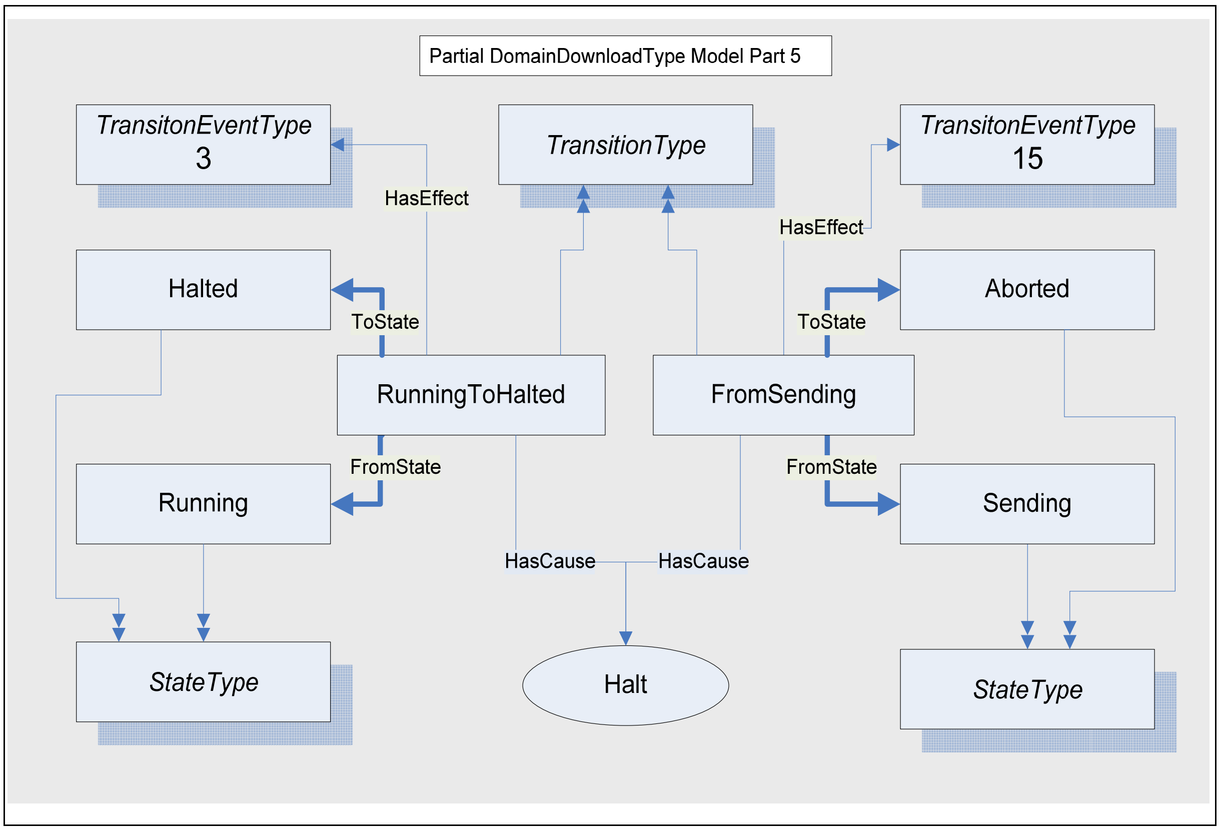

Figure 14 - Running To Halted - Aborted Model

Figure 15 illustrates the model for the Suspended To Halted state transition for an abnormal termination of the domain download. The cause for this transition is the *Halt Method.* The *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.

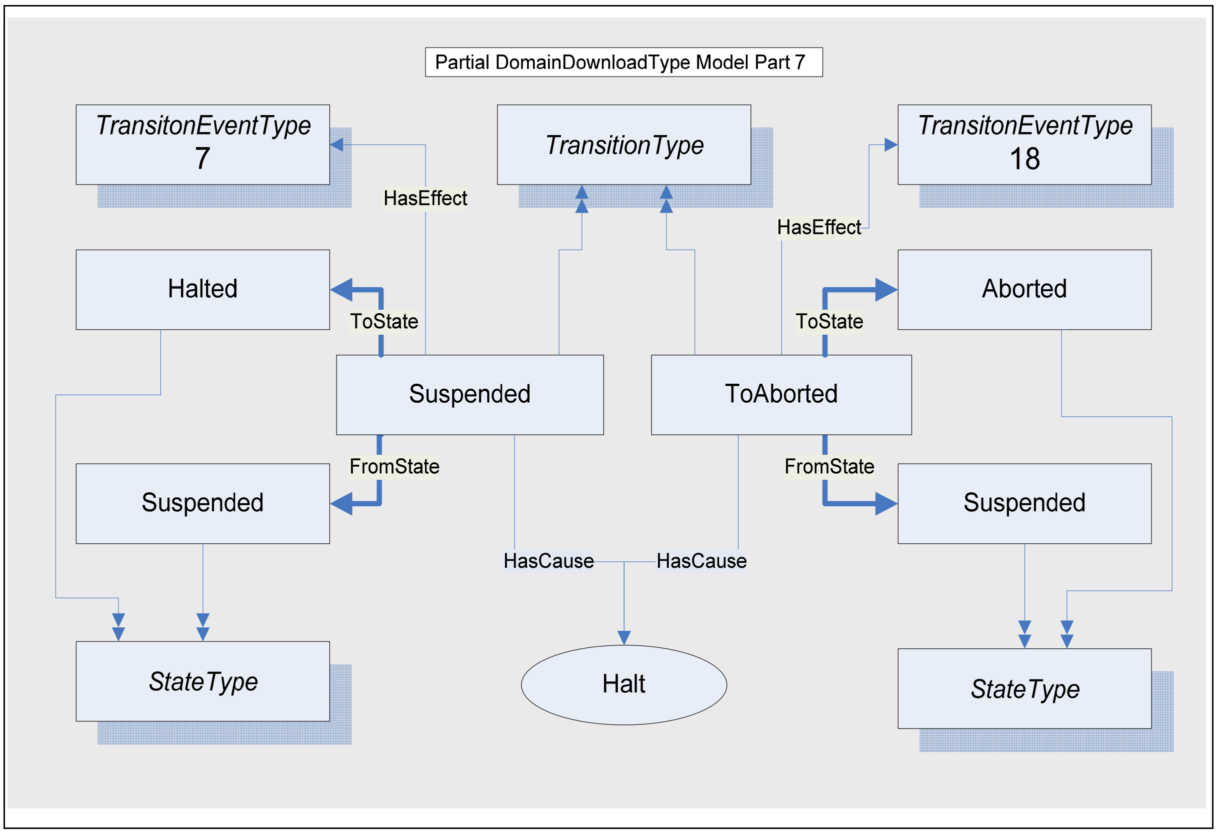

Figure 15 - Suspended To Aborted Model

Figure 16 illustrates the model for the Running To Completed state transition for a normal termination of the domain download. 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 UA Clients who are not subscribed to event notifications.

Table 25 - 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	New
HasComponent	Variable	FailureDetails	Long	VariableType	New

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

Table 26 - Final Result Variables

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

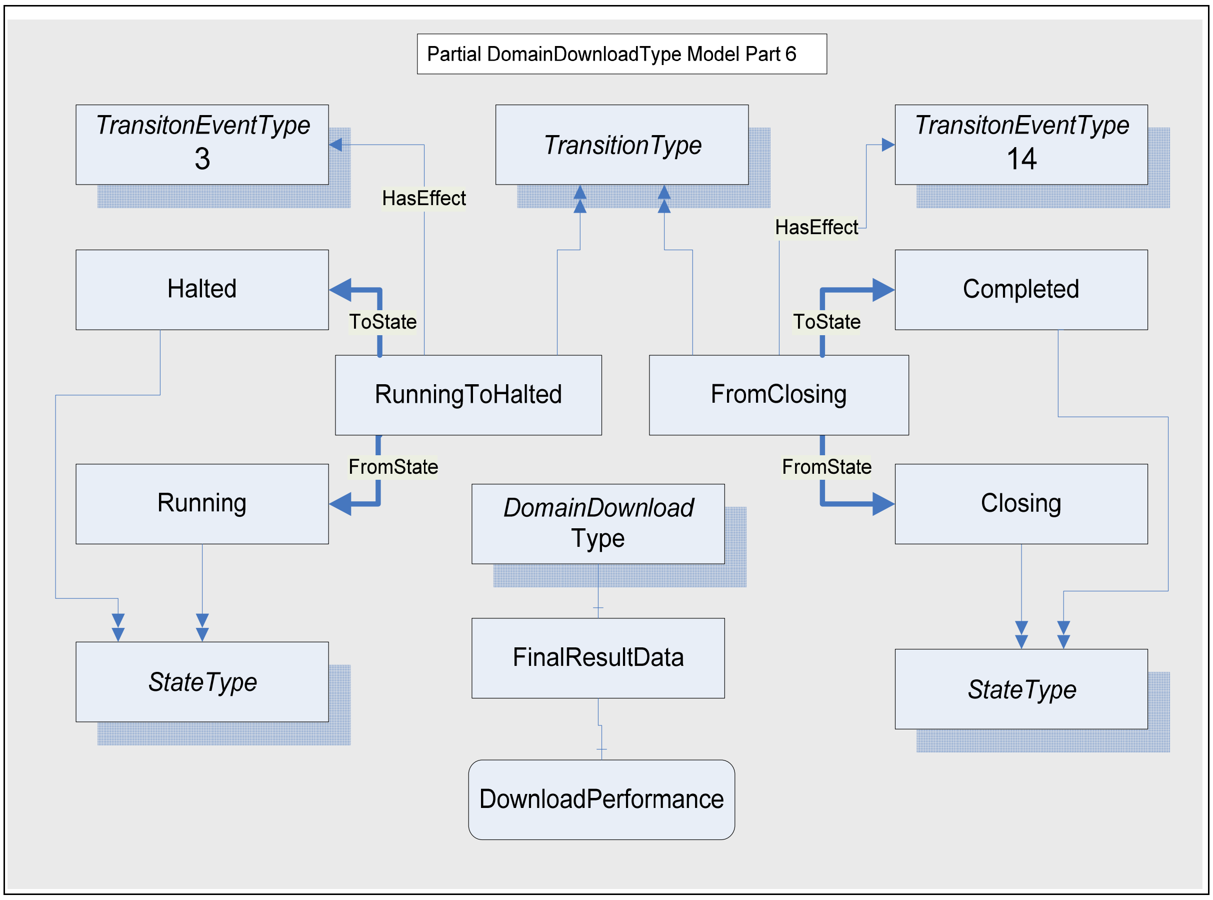

Figure 16 - Running To Completed Model

A.2.5.3 Sequence of Operations

Figure 17 illustrates a normal sequence of service exchanges between a *UA Client* and *UA Server* that would occur during the life cycle of a DomainDownloadType *Program Invocation*.

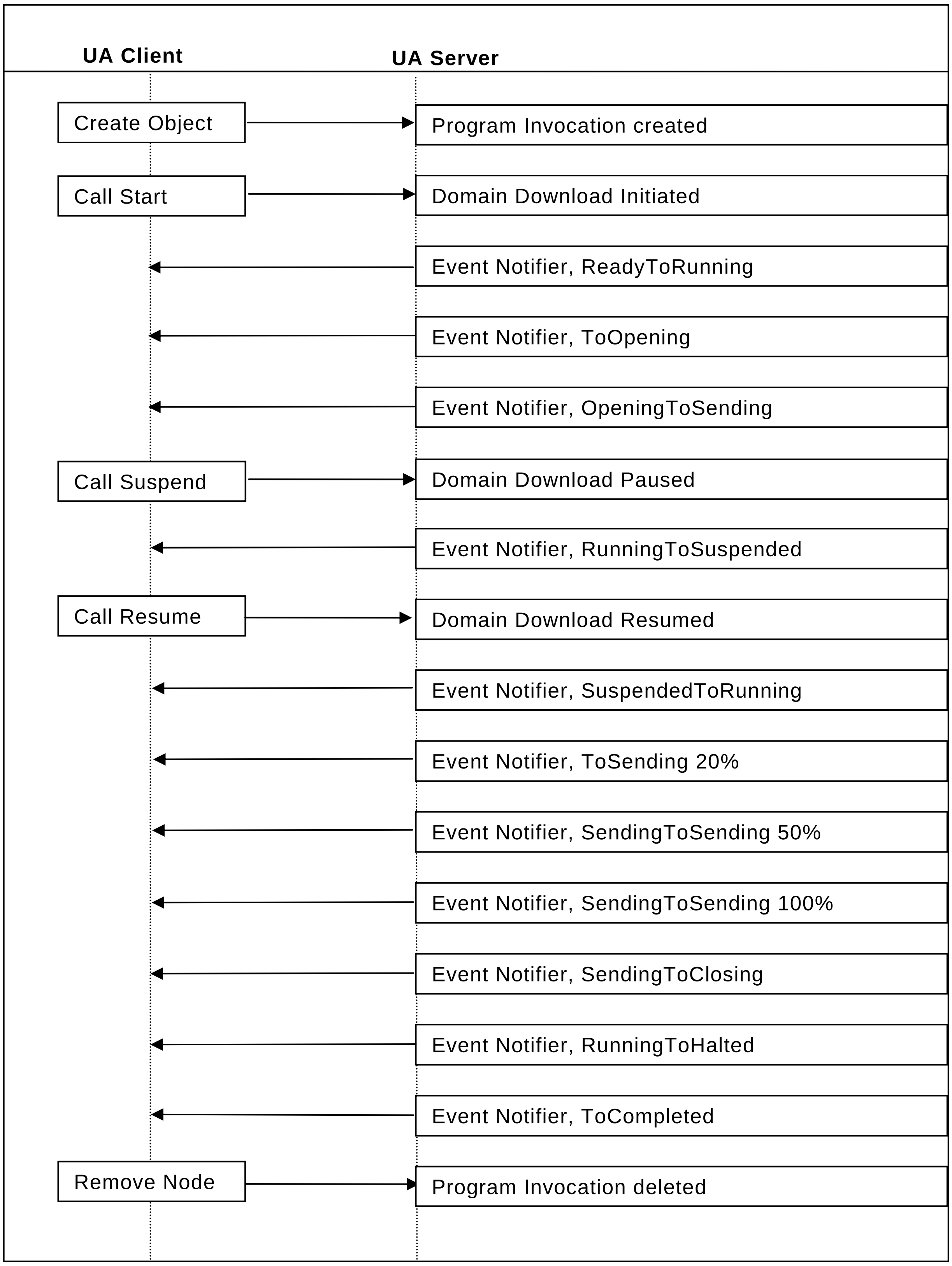
UA Client
UA Server
Create Object
Program Invocation created
Call Start
Domain Download Initiated
Event Notifier, ReadyToRunning
Event Notifier, ToOpening
Event Notifier, OpeningToSending
Call Suspend
Domain Download Paused
Event Notifier, RunningToSuspended
Call Resume
Domain Download Resumed
Event Notifier, SuspendedToRunning
Event Notifier, ToSending 20%
Event Notifier, SendingToSending 50%
Event Notifier, SendingToSending 100%
Event Notifier, SendingToClosing
Event Notifier, RunningToHalted
Event Notifier, ToCompleted
Remove Node
Program Invocation deleted

Figure 17 - Sequence of Operations

Final result Data