

OPC Unified Architecture

Release Candidate Specification

Part 5: SM Appendix

Version 1.70

November 18, 2006

Send comments to:  
[UAcomments@opcfoundation.org](mailto:UAcomments@opcfoundation.org)

|  |  |  |  |
| --- | --- | --- | --- |
| Specification Type | Industry Standard Specification |  |  |
|  |  |  |  |
| Title: | OPC Unified Architecture  SM Appendix | Date: | November 18, 2006 |
|  |  |  |  |
| Version: | Release Candidate 1.70 | Software | MS-Word |
|  |  | Source: |  |
|  |  |  |  |
| Author: | OPC Foundation | Status: | Release Candidate |
|  |  |  |  |

CONTENTS

Page

[Appendix A : StateMachines 1](#__RefHeading___Toc153338084)

[A.1 Scope 1](#__RefHeading___Toc153338085)

[A.2 Examples of finite state machines 1](#__RefHeading___Toc153338086)

[A.2.1 Simple state machine 1](#__RefHeading___Toc153338087)

[A.2.2 State machine containing substates 2](#__RefHeading___Toc153338088)

[A.3 Definition of finite state machine 2](#__RefHeading___Toc153338089)

[A.4 Representation of finite state machines in the AddressSpace 3](#__RefHeading___Toc153338090)

[A.4.1 StateMachineType 3](#__RefHeading___Toc153338091)

[A.4.2 StateType 4](#__RefHeading___Toc153338092)

[A.4.3 InitialStateType 5](#__RefHeading___Toc153338093)

[A.4.4 TransitionType 5](#__RefHeading___Toc153338094)

[A.4.5 FromState 5](#__RefHeading___Toc153338095)

[A.4.6 ToState 6](#__RefHeading___Toc153338096)

[A.4.7 HasCause 6](#__RefHeading___Toc153338097)

[A.4.8 HasEffect 7](#__RefHeading___Toc153338098)

[A.4.9 HasSubStateMachine 7](#__RefHeading___Toc153338099)

[A.4.10 TransitionEventType 8](#__RefHeading___Toc153338100)

[A.5 Examples of StateMachines in the AddressSpace 9](#__RefHeading___Toc153338101)

[A.5.1 StateMachineType using inheritance 9](#__RefHeading___Toc153338102)

[A.5.2 StateMachineType with a sub-machine using inheritance 10](#__RefHeading___Toc153338103)

[A.5.3 StateMachineType using containment 11](#__RefHeading___Toc153338104)

[A.5.4 Alternative approach of StateMachineType using containment 12](#__RefHeading___Toc153338105)

[A.5.5 Example of a StateMachine having Transitions to sub-machines 13](#__RefHeading___Toc153338106)

FIGURES

[Figure 1 – Example of a simple state machine 1](#__RefHeading___Toc153338107)

[Figure 2 – Example of a state machine having a sub-machine 2](#__RefHeading___Toc153338108)

[Figure 3 – Example of a StateMachineType using inheritance 9](#__RefHeading___Toc153338109)

[Figure 4 – Example of a StateMachineType with a sub-machine using inheritance 10](#__RefHeading___Toc153338110)

[Figure 5 – Example of a StateMachineType using containment 11](#__RefHeading___Toc153338111)

[Figure 6 – Example of a StateMachineType using containment (alternative) 12](#__RefHeading___Toc153338112)

[Figure 7 – Example of a state machine with transitions from sub-states 13](#__RefHeading___Toc153338113)

[Figure 8 – Example of a StateMachineType having Transitions to sub-machines 14](#__RefHeading___Toc153338114)

TABLES

Table 1 – StateMachineType Definition [3](#__RefHeading___Toc153338115)

Table 2 – StateType Definition [5](#__RefHeading___Toc153338116)

Table 3 – InitialStateType Definition [5](#__RefHeading___Toc153338117)

Table 4 – TransitionType Definition [5](#__RefHeading___Toc153338118)

Table 5 – FromState ReferenceType [6](#__RefHeading___Toc153338119)

Table 6 – ToState ReferenceType [6](#__RefHeading___Toc153338120)

Table 7 – HasCause ReferenceType [7](#__RefHeading___Toc153338121)

Table 8 – HasEffect ReferenceType [7](#__RefHeading___Toc153338122)

Table 9 – HasSubStateMachine ReferenceType [8](#__RefHeading___Toc153338123)

Table 10 - TransitionEventType [8](#__RefHeading___Toc153338124)

OPC Foundation

\_\_\_\_\_\_\_\_\_\_\_\_

UNIFIED ARCHITECTURE –

FOREWORD

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

Trademarks

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

NON-EXCLUSIVE LICENSE AGREEMENT

The OPC Foundation, a non-profit corporation (the “OPC Foundation”), has defined a set of standard objects, interfaces and behaviour s associated with the objects intended to promote interoperability between automation/control applications, field systems/devices, and business/office applications in the process control industry.

The OPC specifications, sample software that demonstrates the implementation of the specifications, standard interface components deliverables and related documentation (collectively, the “OPC Materials”), form a set of standard objects, interfaces and behaviour that are based on the technology being used in the automation marketplace, and includes the use of Microsoft Technology as well providing interoperability to non Microsoft platforms. The technology defines standard objects, methods, and properties for servers of real-time information like distributed process systems, programmable logic controllers, smart field devices and analyzers in order to communicate the information that such servers contain to standard compliant technologies enabled devices (e.g., servers, applications, etc.).

The OPC Foundation will grant to you (the “User”), whether an individual or legal entity, a license to use, and provide User with a copy of, the current version of the OPC Materials so long as User abides by the terms contained in this Non-Exclusive License Agreement (“Agreement”). If User does not agree to the terms and conditions contained in this Agreement, the OPC Materials may not be used, and all copies (in all formats) of such materials in User’s possession must either be destroyed or returned to the OPC Foundation. By using the OPC Materials, User (including any employees and agents of User) agrees to be bound by the terms of this Agreement.

LICENSE GRANT:

Subject to the terms and conditions of this Agreement, the OPC Foundation hereby grants to User a non-exclusive, royalty-free, limited license to use, copy, display and distribute the OPC Materials in order to make, use, sell or otherwise distribute any products and/or product literature that are compliant with the standards included in the OPC Materials.

All copies of the OPC Materials made and/or distributed by User must include all copyright and other proprietary rights notices include on or in the copy of such materials provided to User by the OPC Foundation.

The OPC Foundation shall retain all right, title and interest (including, without limitation, the copyrights) in the OPC Materials, subject to the limited license granted to User under this Agreement.

WARRANTY AND LIABILITY DISCLAIMERS:

User acknowledges that the OPC Foundation has provided the OPC Materials for informational purposes only in order to help User understand Microsoft’s OLE/COM technology. THE OPC MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. USER BEARS ALL RISK RELATING TO QUALITY, DESIGN, USE AND PERFORMANCE OF THE OPC MATERIALS. The OPC Foundation and its members do not warrant that the OPC Materials, their design or their use will meet User’s requirements, operate without interruption or be error free.

IN NO EVENT SHALL THE OPC FOUNDATION, ITS MEMBERS, OR ANY THIRD PARTY BE LIABLE FOR ANY COSTS, EXPENSES, LOSSES, DAMAGES (INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL OR PUNITIVE DAMAGES) OR INJURIES INCURRED BY USER OR ANY THIRD PARTY AS A RESULT OF THIS AGREEMENT OR ANY USE OF THE OPC MATERIALS.

GENERAL PROVISIONS:

This Agreement and User’s license to the OPC Materials shall be terminated (a) by User ceasing all use of the OPC Materials, (b) by User obtaining a superseding version of the OPC Materials, or (c) by the OPC Foundation, at its option, if User commits a material breach hereof. Upon any termination of this Agreement, User shall immediately cease all use of the OPC Materials, destroy all copies thereof then in its possession and take such other actions as the OPC Foundation may reasonably request to ensure that no copies of the OPC Materials licensed under this Agreement remain in its possession.

User shall not export or re-export the OPC Materials or any product produced directly by the use thereof to any person or destination that is not authorized to receive them under the export control laws and regulations of the United States.

The Software and Documentation are provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation,. 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, the OPC Materials.

1. : StateMachines
   1. Scope

This Appendix describes the basic infrastructure to model finite state machines in OPC UA. It defines *ObjectTypes*, *VariableTypes* and *ReferenceTypes* and explains how they should be used.

This Appendix is normative, i.e. the types defined in the Appendix have to be used as defined. However, it is not required but strongly recommended that a server uses these types to expose its state machines. The defined types may be subtyped to refine their behaviour.

The scope of the state machines described in this Appendix is to provide an appropriate foundation for state machines needed by [UA Part 9] and [UA Part 10]. It does not provide more complex functionality of a state machine like parallel states, forks and joins, history states, choices and junctions etc. However, the base state machine defined in this Appendix can be extended to support such concepts.

The following clauses describe examples of state machines, define state machines in the context of this Appendix and define the representation of state machines in OPC UA. Finally, some examples of state machines, represented in OPC UA, are given.

* 1. Examples of finite state machines
     1. Simple state machine

The following example gives an overview over the base features that the state machines defined in this Appendix will support. In the following clause a more complex example is given, that also supports sub-machines.

Figure 1 gives an overview over a simple state machine. It contains the three states “State1”, “State2” and “State3”. There are transitions from “State1” to “State2”, “State2” to “State2”, etc. Some of the transitions give additional information what causes (or triggers) the transition, e.g. the call of “Method1” for the transition from “State1” to “State2”. The effect (or action) of the transition can also be specified, e.g. the generation of an *Event* of the “EventType1” in the same transition. The notation of the cause is just writing it to the transition, the effect is prefixed with a “/”. More than one cause or effect are separated by a “,”. Not every transition has to have a cause or effect, for example the transition between “State2” and “State3”.



Figure 1 – Example of a simple state machine

For simplicity, the state machines described in this Appendix will only support causes in form of specifying *Methods* that have to be called and effects in form of *EventTypes* of *Events* that are generated, if the transition is used. However, the defined infrastructure allows extending this supporting different causes and effects, if needed.

* + 1. State machine containing substates

Figure 2 shows an example of a state machine where “State6” is a sub-machine. This means, that when the overall state machine is in State6, this state can be distinguish to be in the sub-states “State7” or “State8”. Sub-machines can be nested, i.e. “State7” could be another sub-machine.



Figure 2 – Example of a state machine having a sub-machine

* 1. Definition of finite state machine

The infrastructure of state machines defined in this Appendix only deals with the basic of state machines needed to support [UA Part 9] and [UA Part 10]. The intention is to keep the basic simple but extensible.

For the state machines defined in this Appendix we assume state machines that are typed and where instances of a type cannot change there state machine. However, the base concepts defined in this Appendix can also be used to define more flexible state machines.

Therefore we distinguish between *StateMachineType* and *StateMachine*. The *StateMachineType* contains the description of the state machine – its states, transitions, etc. – whereas the *StateMachine* is an instance of the *StateMachineType* and only contains the current state.

Each *StateMachine* contains information about the current state. If the *StateMachineType* has sub-machines, the *StateMachine* also contains information about the current state of the sub-machines.

Each *StateMachineType* has one or more *States*. For simplicity we do not distinguish between different *States* like the start or the end states.

Each *State* can have one or more sub-machines by referencing a *StateMachine*.

Each *StateMachineType* has one or more *Transitions*. A Transition is directed and points from one *State* to another *State*.

Each *Transition* can have one or more *Causes*. A *Cause* leads a *StateMachine* to change its current State from the source of the *Transition* to its target. In this Appendix we only specify *Method* calls to be *Causes* of *Transitions*.

Each *Transition* can have one or more *Effects*. An *Effect* occurs if the *Transition* is used to change the *State* of a *StateMachine*. In this Appendix we only specify the generation of *Events* to be *Effects* of a *Transition*.

Although this Appendix only specifies simple concepts for state machines, the provided infrastructure is extensible. If needed, special *States* can be defined as well as additional *Causes* or *Effects*.

* 1. Representation of finite state machines in the AddressSpace
     1. StateMachineType

A StateMachineType is represented as a special *ObjectType* in the *AddressSpace* formally defined in Table 1. There are two possibilities to use the base *StateMachineType*: inheritance or containment. If the main reason of an *Object* in the *AddressSpace* is to provide a state machine, its *ObjectType* should inherit from the *StateMachineType*. If the main purpose is not to provide a state machine then the *Object* should contain another *Object* representing the state machine. An example of the first case are Programs defined in [UA Part 10], an example for the latter case are Conditions defined in [UA Part 9].

Table 1 – StateMachineType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | | |
| BrowseName | StateMachineType | | | | | |
| IsAbstract | False | | | | | |
| **References** | | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| Subtype of the BaseObjectType defined in Clause 6.2 of [UA Part 5]. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType. | | | | | | |
| HasComponent | | Variable | CurrentStateBrowseName | QualifiedName | BaseDataVariableType | New |
| HasComponent | | Variable | CurrentStateDisplayName | LocalizedText | BaseDataVariableType | New |
| HasComponent | | Variable | CurrentStateNumber | UInt32 | BaseDataVariableType | New |
| HasComponent | | Variable | CurrentCollapsedStateDisplayName | LocalizedText | BaseDataVariableType | New |
| HasComponent | | Variable | LastTransitionBrowseName | QualifiedName | BaseDataVariableType | New |
| HasComponent | | Variable | LastTransitionDislayName | LocalizedText | BaseDataVariableType | New |
| HasComponent | | Variable | LastTransitionNumber | UInt32 | BaseDataVariableType | New |
| HasComponent | | Variable | TimeOfLastTransition | UtcTime | BaseDataVariableType | New |

The *CurrentStateBrowseName* *Variable* is used for instances of this type to point to its current state. It does not consider sub-states. The state of the sub-machines can be gained from accessing the according *Variable* of the sub-machine.

The *CurrentStateDisplayName* *Variable* contains the *DisplayName* of the *State* *Object* representing the current state. As an example, this *Variable* can be used for user interfaces displaying the current state.

The *CurrentStateDisplayName* *Variable* must always have the value of the *DisplayName* *Attribute* of the *State* that is referenced by the *CurrentStateBrowseName* *Variable.*

The *CurrentStateNumber* *Variable* contains the *StateNumber* of the current state. Each *State* *Object* contains a *StateNumber* that must be unique in the context of *States* directly contained in a *StateMachine* – sub-machinesmay use the same numbers so their *States* cannot be identified that way. As an example the *CurrentStateNumber* can be used to programmatically handle states. Typically it is easier to handle integer values than strings in that case.

The *CurrentStateNumber* *Variable* must always have the value of the *StateNumber* *Variable* of the *State* that is referenced by the *CurrentStateBrowseName* *Variable*.

The *CurrentCollapsedStateDisplayName* *Variable* contains the *DisplayName* of a *State* that is the current state of the StateMachine or one of its sub-machines. There is no rule specified which state or sub-state should be taken. It is up to the server and probably depends on the concrete *StateMachineType* which state to choose. However, if there are no sub-machines for the current state then the value has to be the same as for the *CurrentStateDisplayName* *Variable*. The intention is that clients will use this *Variable* if they display the current state considering sub-state.

The *LastTransitionBrowseName* *Variable* is used for instances of this type to point to the last *Transition* that was fired. It does not consider sub-states. The *Transition* of the sub-machines can be gained from accessing the according *Variable* of the sub-machine. If no *Transition* was used since the *StateMachine* was created, the value must be set to null.

The *LastTransitionDisplayName* *Variable* contains the *DisplayName* of the *Transition* *Object* representing the last *Transition* that was fired. As an example, this *Variable* can be used for user interfaces displaying the last *Transition*.

The *LastTransitionDisplayName* *Variable* must always have the value of the *DisplayName* *Attribute* of the *Transition* that is referenced by the *LastTransitionBrowseName* *Variable.*

The *LastTransitionNumber* *Variable* identifies the *TansitionNumber* of the last *Transition* that was fired. Each *Transition* *Object* contains a *TransitionNumber* that must be unique in the context of *Transitions* directly contained in a StateMachine – sub-machines may use the same numbers so their *Transitions* cannot be identified that way. If no *Transition* was used since the *StateMachine* was created, the value must be set to null.

The *TimeOfLastTransition* *Variable* indicates the time of the last *Transition* that was fired. It does not consider sub-machines. The time of the last *Transition* of a sub-machine can be gained form accessing the according *Variable* of the sub-machine. If no *Transition* was used since the *StateMachine* was created, the value must be set to null.

All *States* and *Transitions* of a state machine are bound to it using *HasComponent* *References*. However, the base *StateMachineType* does not contain any *States* or *Transitions*.

All sub-machines are also bound to the state machine using *HasComponent* *References*, but the base *StateMachineType* does not contain any sub-machines.

*StateMachines* can be defined on two levels. Either by subtyping the *StateMachineType* and thereby defining reusable *StateMachineTypes* or by using the base *StateMachineType* as *InstanceDeclaration* of another *ObjectType* and adding *States* and *Transitions* on this *InstanceDeclaration*. Examples for both approaches are given in Clause A.5.

* + 1. StateType

*States* of a *StateMachine* are represented as *Objects* of the *ObjectType* StateType formally defined in Table 2.

Table 2 – StateType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | | |
| BrowseName | StateType | | | | | |
| IsAbstract | False | | | | | |
| **References** | | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the BaseObjectType defined in Clause 6.2 of [UA Part 5]. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType. | | | | | | |
| HasProperty | | Variable | StateNumber | UInt32 | PropertyType | New |
| HasSubtype | | ObjectType | InitialStateType |  |  |  |

* + 1. InitialStateType

The *InitalStateType* is a subtype of the *StateType* and is formally defined in Table 2. An Object of the *InitialStateType* represents the initial *State* of a *StateMachine*, i.e. the *State* that will initially be entered. Each *StateMachine* can have at most one *State* of type *InitialStateType*, but a *StateMachine* does not have to have a *State* of this type.

Table 3 – InitialStateType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | | |
| BrowseName | InitialStateType | | | | | |
| IsAbstract | False | | | | | |
| **References** | | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Inherit the *Properties* of the *StateType* defined in Clause A.4.2 | | | | | | |

* + 1. TransitionType

*Transitions* of a *StateMachine* are represented as *Objects* of the *ObjectType* *TransitionType* formally defined in Table 4.

Each valid *Transition* must have exactly one *FromState* *Reference* and exactly one *ToState* *Reference*, each pointing to an *Object* of the *ObjectType* *StateType*.

Each *Transition* can have one or more *HasCause* *References* pointing to the cause that triggers the *Transition*.

Each *Transition* can have one or more *HasEffect* *References* pointing to the effects that occur when the *Transition* was triggered.

Table 4 – TransitionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | | |
| BrowseName | TransitionType | | | | | |
| IsAbstract | False | | | | | |
| **References** | | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the BaseObjectType defined in Clause 6.2 of [UA Part 5]. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType. | | | | | | |
| HasProperty | | Variable | TransitionNumber | UInt32 | PropertyType | New |

* + 1. FromState

The *FromState* *ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the starting *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *StateType* or one of its subtypes.

The representation of the *FromState* *ReferenceType* in the *AddressSpace* is specified in Table 5.

Table 5 – FromState ReferenceType

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Value** | | |
| BrowseName | FromState | | |
| InverseName | ToTransition | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
|  |  |  |  |

* + 1. ToState

The *ToState* *ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the ending *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *StateType* or one of its subtypes.

*References* of this *ReferenceType* may be only exposed uni-directional. Sometimes this is required, for example, if a *Transition* points to a *State* of a sub-machine.

The representation of the *ToState* *ReferenceType* in the *AddressSpace* is specified in Table 6.

Table 6 – ToState ReferenceType

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Value** | | |
| BrowseName | ToState | | |
| InverseName | FromTransition | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
|  |  |  |  |

* + 1. HasCause

The *HasCause* *ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to something that causes the *Transition*. In this Appendix we only define *Methods* as *Causes*. However, the *ReferenceType* is not restricted to point to *Methods*.

The *SourceNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasCause* *ReferenceType* in the *AddressSpace* is specified in Table 7.

Table 7 – HasCause ReferenceType

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Value** | | |
| BrowseName | HasCause | | |
| InverseName | MayBeCausedBy | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
|  |  |  |  |

* + 1. HasEffect

The *HasEffect ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to something that will be effected when the *Transition* is triggered. In this Appendix we only define *EventTypes* as *Effects*. However, the *ReferenceType* is not restricted to point to *EventTypes*.

The *SourceNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasEffect* *ReferenceType* in the *AddressSpace* is specified in Table 8.

Table 8 – HasEffect ReferenceType

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Value** | | |
| BrowseName | HasEffect | | |
| InverseName | MayBeEffectedBy | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
|  |  |  |  |

* + 1. HasSubStateMachine

The *HasSubStateMachine ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point from a *State* to a *StateMachine* containing the sub-states of the *State*.

The *SourceNode* of this *ReferenceType* must be an *Object* of the *ObjectType* *StateType* or one of its subtypes. The *TargetNode* must be an *Object* of the *ObjectType* *StateMachineType*. Each *Object* can be the *TargetNode* of at most one *HasSubStateMachine* *Reference*.

The *SourceNode* (the State) and the *TargetNode* (the sub-machine) must belong to the same *StateMachine*, i.e. both must be referenced from the same *Object* of type *StateMachineType* using a *HasComponent* *Reference* or a subtype of *HasComponent*.

The representation of the *HasSubStateMachine* *ReferenceType* in the *AddressSpace* is specified in Table 9.

Table 9 – HasSubStateMachine ReferenceType

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Value** | | |
| BrowseName | HasSubStateMachine | | |
| InverseName | SubStateMachineOf | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
|  |  |  |  |

* + 1. TransitionEventType

The *TransitionEventType* is a subtype of the *BaseEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table 10.

Table 10 - TransitionEventType

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | TransitionEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Inherit the *Properties* of the base *BaseEventType* defined in Clause 6.4.2 | | | | | |
| HasProperty | Variable | TransitionNumber | UInt32 | PropertyType | New |
| HasProperty | Variable | FromStateNumber | UInt32 | PropertyType | New |
| HasProperty | Variable | FromStateNumberSubMachinePath | String | PropertyType | New |
| HasProperty | Variable | ToStateNumber | Uint32 | PropertyType | New |
| HasProperty | Variable | ToStateNumberSubMachinePath | String | PropertyType | New |

The *TransitionEventType* inherits the *Properties* of the *BaseEventType*.

The inherited *Property* *SourceNode* must be filled with the *NodeId* of the *StateMachine* instance were the *Transition* occurs. If the *Transition* occurs in a sub-machine, then the *NodeId* of the sub-machine has to be used. If the Transition occurs between a *StateMachine* and a sub-machine, then the *NodeId* of *theStateMachine* has to be used, independent of the direction of the *Transition*.

The *TransitionNumber* *Property* identifies the *Transition* that triggered the *Event*.

The FromStateNumber *Property* identifies the *StateNumber* of the originating *State* of the *Transition*.

Since the *Transition* could start from a *State* of a sub-machine the *FromStateNumber* *Property* is ambiguous. Therefore the *Property* *FromStateNumberSubMachinePath* identifies the path to the sub-machine containing the state starting from the *NodeId* available in the *SourceNode* *Property*. If the *State* is not in a sub-machine, *ToStateNumberSubMachinePath* is set to null. The String uses the syntax of the *TranslateBrowsePathsToNodeIds* service defined in [UA Part 4] to specify the path.

The *ToStateNumber* *Property* identifies the *StateNumber* of the terminal *State* in the *Transition*.

Since the *Transition* could point to a *State* of a sub-machine the *ToStateNumber* *Property* is ambiguous. Therefore the *Property* *ToStateNumberSubMachinePath* identifies the path to the sub-machine containing the state starting from the *NodeId* available in the *SourceNode* *Property*. If the *State* is not in a sub-machine, *ToStateNumberSubMachinePath* is set to null. The String uses the syntax of the *TranslateBrowsePathsToNodeIds* service defined in [UA Part 4] to specify the path.

* 1. Examples of StateMachines in the AddressSpace
     1. StateMachineType using inheritance



Figure 3 – Example of a StateMachineType using inheritance

In Figure 3 an example of a StateMachine is given using the Notation defined in Appendix xxx. First, a new state machine type is defined, called “MyStateMachineType”, inheriting from the base StateMachineType. It contains two States, “State1” and “State2” and a *Transition* “Transition1” between them. The *Transition* points to a *Method* “MyMethod” as the cause of the *Transition* and an *EventType* “EventType1” as the effect of the *Transition*.

Instances of “MyStateMachineType” can be created, for example “MyStateMachine”. It has a *Variable* “CurrentStateNumber” representing the current state. Note that the other *Variables* representing the current state are omitted in the Figure for simplicity. The “MyStateMachine” *Object* directly references the information of the StateMachine – *States* and *Transitions* – since they have the *ModellingRule* “Shared” (not shown in the Figure). In the example this is also true for “MyMethod” – other StateMachines may have non-shared Methods.

* + 1. StateMachineType with a sub-machine using inheritance



Figure 4 – Example of a StateMachineType with a sub-machine using inheritance

Figure 4 gives an example of a StateMachineType having a sub-machine for its “State1”. For simplicity no effects and causes are shown, as well as type information for the states or *ModellingRules*.

The “MyStateMachineType” contains an Object “MySubMachine” of type “AnotherStateMachineType” representing a sub-machine. It has the *ModellingRule* “New” not shown in the Figure. The “State1” references this *Object* with a *HasSubStateMachine* *Reference*, thus it is a sub-machine of “State1”. Since “MySubMachine” is an *Object* of type “AnotherStateMachineType” it references its shared components – in this case only “StateX” and has a *Variable* representing the current state. Since it is used as an *InstanceDeclaration*, no value is assigned to this *Variable*.

An *Object* of “MyStateMachineType”, called “MyStateMachine”, does not only reference the shared components of “MyStateMachineType” and has *Variables* for the current state, but also has an *Object* “MySubMachine” and *Variables* representing the current state of the sub-machine. Since the sub-machine is only used when “MyStateMachine” is in “State1”, a client would get a BAD *StatusCode* assigned for the *Variables* if “MyStateMachine” is in a different state.

* + 1. StateMachineType using containment



Figure 5 – Example of a StateMachineType using containment

Figure 5 gives an example of an *ObjectType* not only representing a StateMachine but also having some other functionality. The *ObjectType* “MyObjectType” has an *Object* “MyComponent” representing this other functionality. But is also contains a StateMachine “MyStateMachine” of the type “MyStateMachineType”. *Objects* of “MyObjectType” also contain such an *Object* representing the StateMachine and a *Variable* containing the current state of the StateMachine, as shown in the Figure.

* + 1. Alternative approach of StateMachineType using containment



Figure 6 – Example of a StateMachineType using containment (alternative)

Figure 6 gives an example of an alternative way of modelling an *ObjectType* not only representing a StateMachine but also having some other functionality. The *ObjectType* “MyObjectType” has an *Object* “MyComponent” representing this other functionality. But is also contains a StateMachine “MyStateMachine” of the base StateMachine type “StateMachineType”. The States are not defined by a subtype of “StateMachineType” but directly as *InstanceDeclaration* of “MyObjectType”. This approach is suitable if it is clear that the StateMachine is only used in the context of “MyObjectType”. The States still have the *ModellingRule* “Shared” and are only references from the *Objects* of type “MyObjectType”.

* + 1. Example of a StateMachine having Transitions to sub-machines

The *StateMachines* shown so far only had *Transitions* between *States* on the same level, i.e. on the same *StateMachine*. Of cause, it is possible and often required to have *Transitions* between *States* of the *StateMachine* and *States* of its sub-machine.

Because a sub-machine can be defined by another *StateMachineType* and this type can be used in several places, it is not possible to add a bi-directional *Reference* from one of the shared *States* of the sub-machine to another *StateMachine*. In this case it is suitable to expose the *FromState* or *ToState* *References* uni-directional, i.e. only pointing from the *Transition* to the *State* and not have the other direction browsable. If a *Transition* points from a *State* of a sub-machine to a *State* of another sub-machine, both, the *FromState* and the *ToState* *Reference*, are handled uni-directional.

A Client must be able to handle the information of a StateMachine if the *ToState* and *FromState* *References* are only exposed as forward *References* and the inverse *References* are omitted.

Figure 7 gives an example of a state machine having a transition from a sub-state to a state.



Figure 7 – Example of a state machine with transitions from sub-states

In Figure 8, the representation of this example as *StateMachineType* in the *AdressSpace* is given. The “Transition1”, part of the definition of “MyStateMachineType”, points to the “StateX” of the *StateMachineType* “AnotherStateMachineType”. The *Reference* is only exposed as forward *Reference* and the inverse *Reference* is omitted. Thus, there is no *Reference* from the “StateX” of “AnotherStateMachineType” to any part of “MyStateMachineType” and “AnotherStateMachineType” can be used in other places as well.



Figure 8 – Example of a StateMachineType having Transitions to sub-machines