Using AddIns
to Enhance Objects and Variables

Whitepaper Version 1.0

September 27, 2011

Randy Armstrong, OPC Foundation

Matthias Damm, ascolab GmbH

Karl-Heinz Deiretsbacher, Siemens AG

Jim Luth, Invensys

Wolfgang Mahnke, ABB

1 Background

OPC UA defines a type model for *Objects* and *Variables* supporting one object-oriented type hierarchy for *Object Types* and another one for *Variable Types* [UA Part 3, UA Part 5]. Although the specification does not restrict those hierarchies to be single inheritance (i.e. a type can only have one super-type) it only specifies the semantic (inheritance rules) for single inheritance. It does not define what would happen with conflicts resulting from multiple inheritance like having two super-types with *Variables* of the same name but different *Type*s (and semantic).

In general, good object-oriented design is accomplished by not over-using inheritance but instead by using composition to aggregate an object providing several functionalities [GH95, FF04]. The same applies for OPC UA models where we recommend the use of composition instead of multiple inheritance.

This Whitepaper describes a 'Best Practice' how to extend *Objects* and *Variables* in OPC UA and avoid multiple inheritances. It takes special characteristics of the OPC UA model into account, like the ability to specify additional properties for standard OPC UA *Object Types*.

This Whitepaper is not only intended for server vendors or information modellers providing an OPC UA information model, but also for those consuming the information model as they can make use of the concepts in their client applications.

2 Model

2.1 Overview

The OPC UA composition concept described in this Whitepaper is called AddIn. There are other ways to achieve composition. Using the AddIn concept assures consistency across different OPC UA Information Models.

The AddIn concept can be used when sub-typing is not suitable for the required extension and the AddIn model covers areas of extension that cannot be solved by simple sub-typing:

- It allows enhancing multiple types at arbitrary positions in the type hierarchy.
- It allows enhancing just instances.

2.2 AddIn Element

AddIn elements most commonly will be modeled as *Object Types*. They are indistinguishable from other *Object Types* used to expose full functionality with one exception: instances of such *Object Types* will always have pre-defined *BrowseNames*.

Figure 1 illustrates two *Object Types* that represent such elements:

- LockService, a concept to lock an object and
- ConfigurableObject, a concept to provide and manage the configuration of an object.

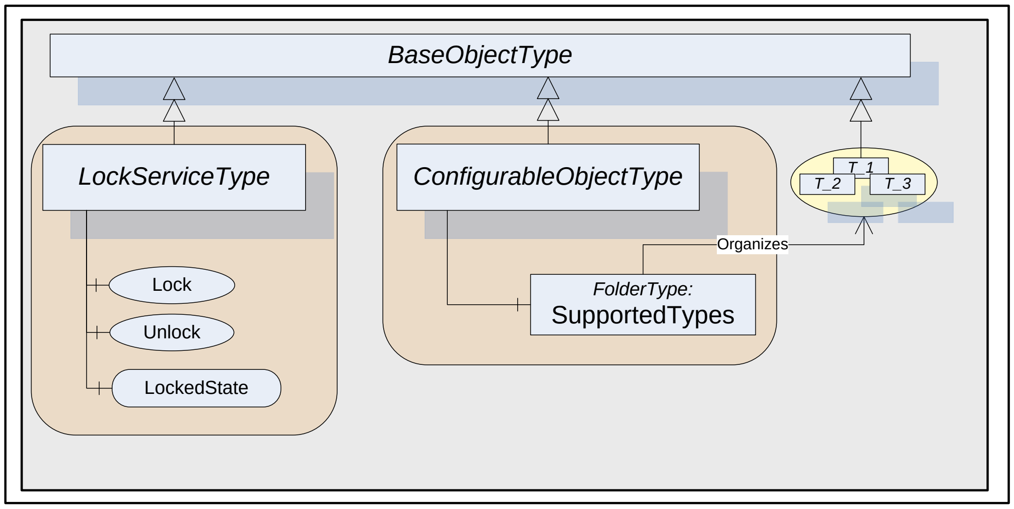

Figure 1 –Examples of Object Types that can be used as AddIns

2.3 Applying AddIns

AddIn elements are added to existing OPC UA Address Space elements (e.g. *Object/Variable* (*Types*)) by aggregation. Typically, a *HasComponent Reference Type* will be used to refer from the existing *AddressSpace* element (e.g. *Object* or *Object Type*) to the added element. However, since *Variables* can not have *Objects* as components [UA Part 3], other *Reference Types* (preferably subtypes of Aggregates) have to be used in this case.

The aggregated feature shall have a pre-defined *BrowseName* in order to identify it as a specific Addin. This is exemplified in Figure 2. The *Object Type* LockServiceType is used as AddIn with the defined *BrowseName* “Locking”. The AddIn Locking provides methods to lock and unlock an object and only users that have locked the object can change the object. The AddIn can already be defined on a *Type* (e.g. XYZ-DeviceType in Figure 2) and thus apply for all instances of that type (e.g. MD002) or be directly used on an instance (e.g. MD001) without being defined on the type.

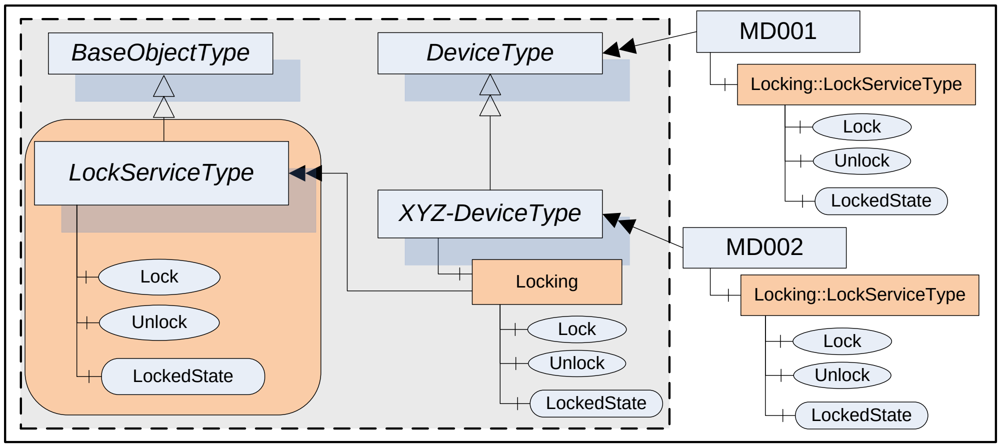

Figure 2 – Use of AddIn BrowseName

Another example of an AddIn used on *Variables* is specified in [UA Part 11] where historical data nodes reference their configuration objects with the *Reference Type* “HasHistoricalConfiguration” as a sub-type of Aggregates and the standardized *BrowseName* “HA Configuration”.

The use of pre-defined *BrowseNames* is an important directive since it facilitates detection and usage of features as specified in the Section 2.5.

2.4 Defining AddIns

As described in the previous section an AddIn can be applied in multiple places: On different *Type*s, on instances of those *Type*s and on instances where the *Type* does not contain the AddIn information. The simplest form of an AddIn is when it is only used on one type and thus the typical usage of an *Instance Declaration* in OPC UA applies. However, the intention of the AddIn described in this Whitepaper is that it is to be used in more than one type definition not in the same branch of the type hierarchy or as an addition on instances of other *Type*s.

One way for defining AddIns is to write the *BrowseName* and the semantic as well as the *Type* used for the AddIn on a “piece of paper” like a vendor user manual or a standard OPC UA information model specification. As the *BrowseName* contains a Namespace Index, it is up to the organization defining the AddIns to assure that it is unique and that no name conflicts occur.

In addition, it is often desirable to make the information about AddIns available in the meta data of the OPC UA server. AddIns that are applicable to all *Objects* can be included as optional *Instance Declaration*s on the *BaseObjectType* (or the *BaseDataVariableType* for *Variables*) as OPC UA allows adding *Instance Declarations* on these base types. If only a part of the *Type* hierarchy can support the AddIn, the most common base type for all those *Type*s should be used.

On specific *Type*s where the AddIn is mandatory for all instances, the type should declare the AddIn as mandatory instead of optional. Note that it is only possible to change the required *Type* for the AddIn to a more specific type when overriding the AddIn.

2.5 Detecting and Using AddIns

Clients can detect the implementation of AddIns by browsing or querying for the *BrowseName* and (optionally) the *Type* of the AddIn.

As a result of the pre-defined *BrowseName*, a single *TranslateBrowsePathToNodeId* request can be used to get the *NodeId* of AddIn components. In the example in Figure 2 the relative path “**Locking/Lock**” with the starting node **MD002** can be used to request the *NodeId* of this *Method* and the relative path “**Locking/LockedState**” can be used for the respective state *Property*.

It is fundamental to AddIn elements that they act upon or for the owning *Object*. As an example, the *Methods* “Lock”, “Unlock” shown in Figure 2 will act on **MD002** although the objectId parameter in the Call *Service* has to contain the *Method* owner – i.e. the *NodeId* of the **Locking** *Object*.

Clients can determine the *BrowseName* for specific AddIns either by some written documentation or by looking at the Instance Declarations of the type system.

3 References

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model 1.01, Feb. 2009

http://www.opcfoundation.org/UA/Part3/

[UA Part 5] OPC UA Specification: Part 5 – Information Model 1.01, Feb. 2009

http://www.opcfoundation.org/UA/Part5/

[UA Part 11] OPC UA Specification: Part 11 – Historical Access RC 1.01.02, Dec. 2009

http://www.opcfoundation.org/UA/Part11/

[GH95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley 1995

[FF04] Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates: Head First Design Patterns, O’Reilly, 2004