Available online at www.sciencedirect.com
ScienceDirect
Procedia CIRP 25 (2014) 297 – 304
www.elsevier.com/locate/procedia

8th International Conference on Digital Enterprise Technology - DET 2014 – "Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution

Interoperability between OPC UA and AutomationML

Robert Henßen, Miriam Schleipen*

Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB), Fraunhoferstr. 1, 76131 Karlsruhe, Germany

* Corresponding author. Tel.: +49-721-6091-382; fax: +49-721-6091-413. *E-mail address:* miriam.schleipen@iosb.fraunhofer.de

Abstract

OPC UA (OPC Unified Architecture) is a platform-independent standard series (IEC 62541) [1], [2] for communication of industrial automation devices and systems. The OPC Unified Architecture is an advanced communication technology for process control. Certainly the launching costs for the initial information model are quite high. AutomationML (Automation Markup Language) is an upcoming open standard series (IEC 62714) [3], [4] for describing production plants or plant components. The goal of this contribution is to simplify the creation of OPC UA information models based on existing AutomationML data by examining the analogies between AutomationML and the OPC UA information model.

Keywords: AutomationML; OPC UA; Mapping

1. Introduction

Engineering information plays an important role for engineering efficiency. [5] This contribution tries to bring the engineering format AutomationML in contact to online production data and extends the application domain of OPC UA. Furthermore, AutomationML engineering data (offline) can be operationalized by means of the online communication in OPC UA. This will be reached by taking advantage of analogies between OPC UA and AutomationML. The OPC UA information model base types will be expanded with AutomationML specific ones and required mapping rules will be drafted. Chapter 2 explains OPC UA and its information model concept. In chapter 3 the authors explain AutomationML and show one small example plant model. Chapter 4 contains analogies between the two standards and proposes necessary type definitions within the OPC UA context and explains these aspects by means of the example plant model. In Chapter 5 the authors will give a short summary and outlook.

2. Online communication in OPC UA

OPC UA specifies the exchange of real-time information of production plant data between control devices or IT systems from different manufacturers. OPC UA server includes an information model that allows users to organize data and their semantics in a structured manner. The information model constitutes the address spaces of OPC UA servers. It is a full-mesh network of nodes with their properties and relations. In general, users create the information model for their OPC UA servers manually [6] at implementation time or implement vendor-specific automatisms. A server address space consists of the following element types:

- Object: "A Node that represents a physical or abstract element of a system. Objects are modelled using the OPC UA Object Model. Systems, subsystems and devices are examples of Objects. An Object may be defined as an instance of an ObjectType." [7]
- ObjectType: "A Node that represents the type definition for an Object." [7]
- Variable: "A Variable is a Node that contains a value." [7]

doi:10.1016/j.procir.2014.10.042

- VariableType: “Node that represents the type definition for a Variable” [8]
- DataType: “An instance of a DataType Node that is used together with the ValueRank Attribute to define the data type of a Variable.” [8]
- ReferenceType: “A Node that represents the type definition of a Reference. The ReferenceType specifies the semantics of a Reference. The name of a ReferenceType identifies how source Nodes are related to target Nodes and generally reflects an operation between the two, such as “A Contains B”.” [7]
- Method: “A callable software function that is a component of an Object.” [7]
- View: “A specific subset of the AddressSpace that is of interest to the Client.” [7]

In this contribution, the authors use some predefined reference types which are explained in the following listing:

- HasComponent: “The semantic is a part-of relationship. The TargetNode of a Reference of the HasComponent ReferenceType is a part of the SourceNode. This ReferenceType is used to relate Objects or ObjectTypes with their containing Objects, DataVariables, and Methods as well as complex Variables or VariableTypes with their DataVariables.” [8]
- HasProperty: “The semantic is to identify the Properties of a Node.” [8]
- HasTypeDefinition: “The semantic of this ReferenceType is to bind an Object or Variable to its ObjectType or VariableType, respectively.” [8]
- HasSubType: “The semantic of this ReferenceType is to express a subtype relationship of types. It is used to span the ReferenceType hierarchy.” [8]

Figure 1 depicts the graphical notation given by the OPCFoundation (see [7]) for modelling address spaces. The graphical notations for the references are depicted in Figure 2. There is also an XML format defined by OPC UA which can be used to describe address spaces.

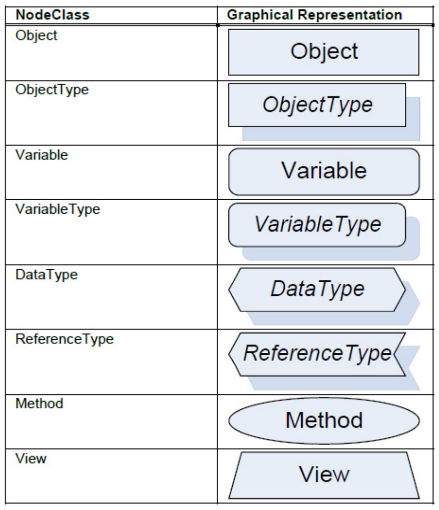

Fig. 1. OPC UA address space element types [7].

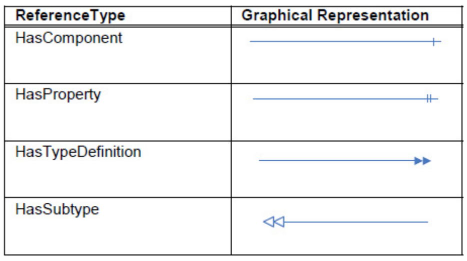

Fig. 2. OPC UA address space reference types [7].

The OPCFoundation provides a set of base types which can be used to create new objects and types derived from these standard types. Some organizations define so called companion specifications, e.g. ‘OPC UA For Devices’ (DI) and ‘OPC UA For Analyzer Devices’ (ADI). “DI is a companion specification designed for exposing and functionally grouping Device Parameters and Methods. ADI is a companion specification for sophisticated Analyzer Devices like Spectrometers or Chromatographs.” [9] Such a companion specification does not already exist for AutomationML. This contribution could be a rough base for such an AutomationML companion specification.

OPC UA is interesting and important for IT systems within the production environment, e.g. MES (Manufacturing Execution System) as data and integration platform, particularly due to its numerous interfaces.

3. Engineering data (offline) in AutomationML

AutomationML is an XML based data format especially designed for the exchange of plant engineering information. The format interconnects engineering tools of different disciplines and lifecycle phases, from plant construction over mechanical and electrical design to virtual start-up. AutomationML doesn’t define a new file format itself. It uses several existing and well-proven standards and defines rules to use and combine them. CAEX (IEC 62424) [10] is used as top level format, i.e. the main AutomationML file is in fact a CAEX file. Other used formats are Collada [11] for geometry and PLCOpenXML [12] for behaviour information. This contribution will focus on the top-level definitions of AutomationML models which are implemented in the format CAEX. CAEX provides object-oriented concepts such as classes and instances, and the possibility to describe arbitrary meshes. Disregarding the last feature the objects are stored in a tree structure.

3.1. Structure of AutomationML

Following, the main structure and objects appearing in AutomationML models will be described. The whole AutomationML structure, the most important elements, and the relations between them are compressed to three figures.

Each AutomationML file can contain several libraries: InterfaceClassLibs for defining interfaces, RoleClassLibs for semantic role definitions and SystemUnitClassLibs which include reusable AutomationML objects. The fourth container type is the InstanceHierarchy which consists of real plant descriptions.

Figure 3 depicts this basic structure of an AutomationML file, the possible container structures (the three library types and the InstanceHierarchy) and for each of this the main including objects. The arrows in this figure depict a 'consists of' relation.

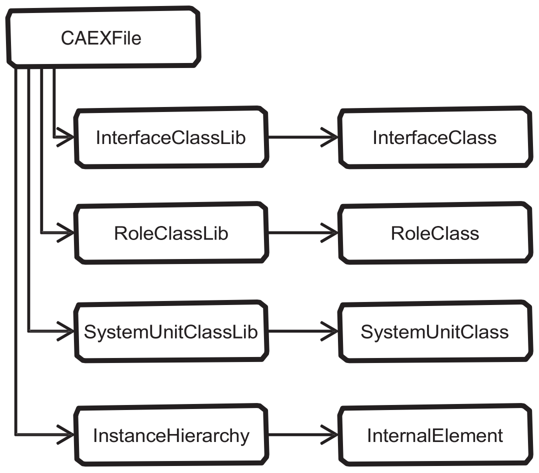

Fig. 3. AutomationML basic file structure.

The main AutomationML elements and all possible relations between them are shown in Figure 4. The organization in a hierarchical tree involves that every element type can have child elements of the same type (see ChildElement relations). SystemUnitClasses and RoleClasses are defined in an inheritance structure; the RefBaseClassPath relations define such inheritance relations. RoleClasses can be assigned to SystemUnitClasses and InternalElements. In the latter case there are two different ways to make the RoleClass assignment using SupportedRoleClasses or the RoleRequirement. The remaining arrows in Figure 4 describe the possible relations between InternalElements and SystemUnitClasses. An InternalElement can inherit from a SystemUnitClass. In the scope of AutomationML SystemUnitClasses are only templates and can be changed after instantiation. The backward relation indicates that a SystemUnitClass can define sub elements in terms of InternalElements. These sub elements (InternalElements) should be included on instantiation.

RoleClasses describe functions of a physical or logical plant object independent of a technical implementation. They offer a possibility to specify an object on an abstract way and independent of the manufacturer. The assignment of a RoleClass to an object (SystemUnitClass or InternalElement) results in the allocation of fundamental functions or requirements.

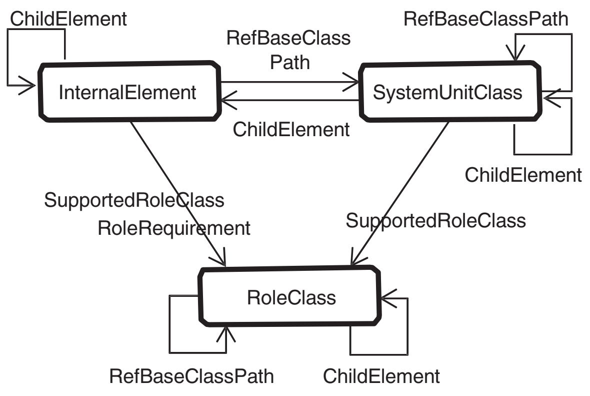

Fig. 4. AutomationML main elements.

Figure 5 depicts the basic structure of a RoleClass, SystemUnitClass, or InternalElement. Each can contain arbitrary nested Attributes and Interfaces (see ChildElement relations). The Interfaces are called ExternalInterfaces and shall inherit from an InterfaceClass. They can be connected to other ExternalElements via InternalLinks. The InterfaceClasses are stored hierarchically and shall have independent inheritance relations. InterfaceClasses and certainly ExternalInterfaces may also have arbitrary nested Attributes.

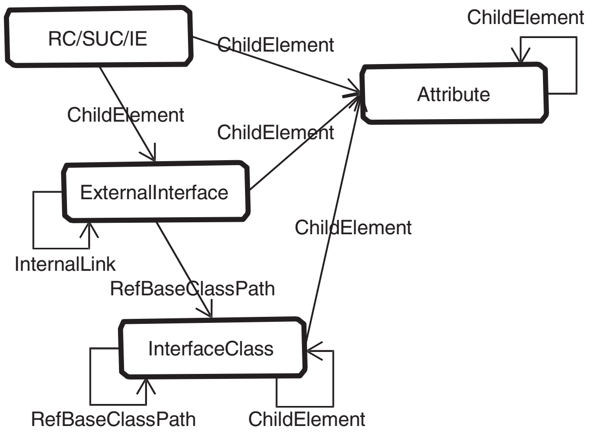

Fig. 5. AutomationML main object details.

In addition to this, AutomationML defines extended concepts and a basic set of semantic roles. The Process-Product-Resource concept allows high level structuring of engineering data based on a process-centric, product-centric, or resource-centric view including relations between them. The AML Port concept allows a high level description of complex interfaces. It makes possible to combine several interfaces to a complex plug. The AML Facet concept allows the storage of a subset of attributes and interfaces at one AML object. The AML Group concept allows the storage of separate views on a subset of the AML objects. The PropertySet concept was defined for assigning semantics to attributes or groups of attributes.

3.2. Example model in AutomationML

Figure 6 depicts the tree structure of an AutomationML example. The example model includes a small InstanceHierarchy called 'TestProject' with a production line including two different robots.

The AutomationMLInterfaceClassLib and the AutomationMLBaseRoleClassLib are slimmed-down versions of the ones defined in the AutomationML standard. They contain only the used elements to shorten the example.

The project-specific RoleClassLib 'TestRoleLib' defines a role 'Robot' which includes an attribute and a communication interface and is derived from the standardised RoleClass 'Resource'. These detail definitions of the role are not visible in Figure 6. The second role 'SpecialRobot' is derived from the first one. The SystemUnitClassLib 'ABCSystemUnitclassLib' specifies example types for a robot (assigned to the role Robot) and a production line.

Back to the starting point, our TestProject: The instances 'MainLine' and the first robot 'RobotI' instantiate the types of the SystemUnitClass without changes. The second robot 'RobotII' has no corresponding object in the SystemUnitClassLib. Its special requirements are expressed by an assignment to the role 'SpecialRobot'.

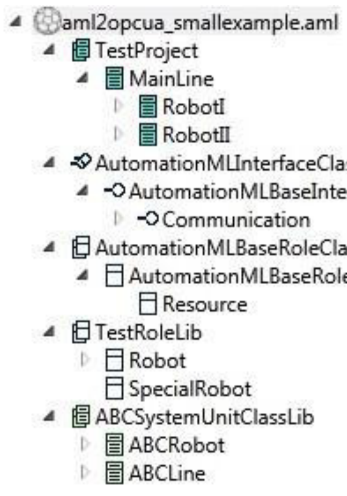
```
aml2opcua_smallexample.aml
  TestProject
    MainLine
      RobotI
      RobotII
  AutomationMLInterfaceClassLib
    AutomationMLBaseInterface
      Communication
  AutomationMLBaseRoleClassLib
    AutomationMLBaseRole
      Resource
  TestRoleLib
    Robot
    SpecialRobot
  ABCSystemUnitClassLib
    ABCRobot
    ABCLine
```

Fig. 6. Object tree of the AutomationMLexample.

4. Creation of OPC UA information models by means of AutomationML models

4.1. Analogies between AutomationML and the OPC UA information model

A short comparison follows to prelude the mapping between the information models of OPC UA and AutomationML. Both standards pursue an object-oriented approach and based on the type-instance concept.

The analogy between UA objects and InternalElements is obvious. But already the inheritance mechanism is handled different. In AutomationML SystemUnitClasses are only templates for the InternalElements. It is explicitly allowed to add and remove parts of the instance. OPC UA handles the instantiation via the HasTypeDefinition reference more explicit, the ModellingRules specify whether the sub nodes have to exist after instantiation. In case of the mapping from InternalElements to UA objects, the ModellingRules of all sub nodes shall be set to 'Optional'. There must be a basic and empty SystemUnitClass type in OPC UA to provide a type for InternalElements without a reference to a SystemUnitClass.

Typecasting is in both standards not limited to objects. They use this principle for interfaces and relations also. Similarly the differentiation between objects and properties (OPC UA) respectively attributes (AutomationML).

By default XML elements are organized in a tree structure alike AutomationML. Certainly AutomationML has methods to describe arbitrary meshes as well as OPC UA.

OPC UA defines the semantics of objects in ObjectTypes. However AutomationML uses RoleClasses to assign semantics to objects not primarily the corresponding type (SystemUnitClass).

4.2. Expansion of OPC UA information model base types with AutomationML specific ones

The main structure of an AutomationML file from Figure 3 is mapped into a slightly variant structure depicted in Figure 7. InstanceHierarchies can be stored at arbitrary positions in an OPC UA server model. The libraries shall be stored in a common folder.

Following the AutomationML specific OPC UA definitions will be listed.

New OPC UA ReferenceType:

- ReferenceType 'HasAMLChild'
  - Subtype of 'HasChild'
  - Hierarchical relation within the AutomationML tree hierarchy.
  - These relations have no special meaning in AutomationML. Maybe there is no need to map these and they can be omitted.

New OPC UA FolderTypes (all of these ObjectTypes are subtypes from the ObjectType 'FolderType'):

- ObjectType 'AMLLibrariesFolderType'
  - Folder to exclusively store AutomationML libraries. Due to the fact that OPC UA allows TargetNodes with same BrowseNames, it is possible to store libraries in different versions at the same time which is forbidden in AutomationML. This offers the possibility to store several AutomationML files in one OPC UA information model.
- ObjectType 'AMLClassLibFolderType'
  - Type for the three AutomationML libraries, all AutomationML libraries shall be mapped to objects of this type.

- Objects of this type are only allowed to have references of the type 'HasAMLChild' with TargetNodes of type CAEXObjectType.

- ObjectType 'AMLInstanzHierarchyFolderType'
  - Type for mapped InstanzHierarchies
  - Objects of this type are only allowed to have references of the type 'HasAMLChild' with TargetNodes of instances of type 'AMLSystemUnitClass'.

Further new OPC UA ObjectTypes:
- ObjectType 'CAEXObjectType'
  - Subtype of 'BaseObjectType'
  - Abstract ObjectType, base for all AutomationMLClasses
- ObjectTypes 'AMLInterfaceClass', 'AMLRoleClass' and 'AMLSystemUnitClass'
  - Subtype of 'CAEXObjectType'
  - These are the basic ObjectTypes for the AutomationMLClasses.

Figure 8 depicts the mapping of the relations between the main AutomationML elements (see Figure 4). The structure of the picture is not changed, so the mapping can be read directly from the graphic.

One new OPC UA type is used:
- ReferenceType 'HasAMLRoleReference'
  - Subtype of 'NonHierarchicalReferences'
  - The SourceNode shall be an object and the TargetNode shall be a subtype of 'AMLRoleClass'.

To use the inheritance possibility of roles it is needed that RoleClasses will be mapped to ObjectTypes. RoleClasses can't be assigned via 'HasTypeDefinition' references or subtypes of this, because OPC UA doesn't support multiple inheritances. The two AutomationML mechanisms SupportedRoleClass and RoleRequirement are handled in the same way.

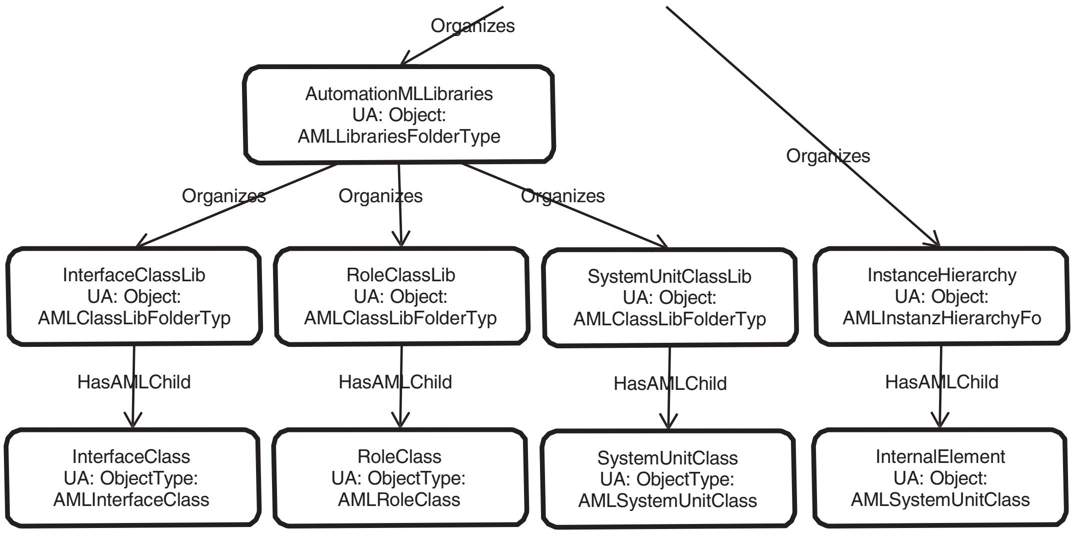

Fig. 7. AutomationML main structures in OPC UA.

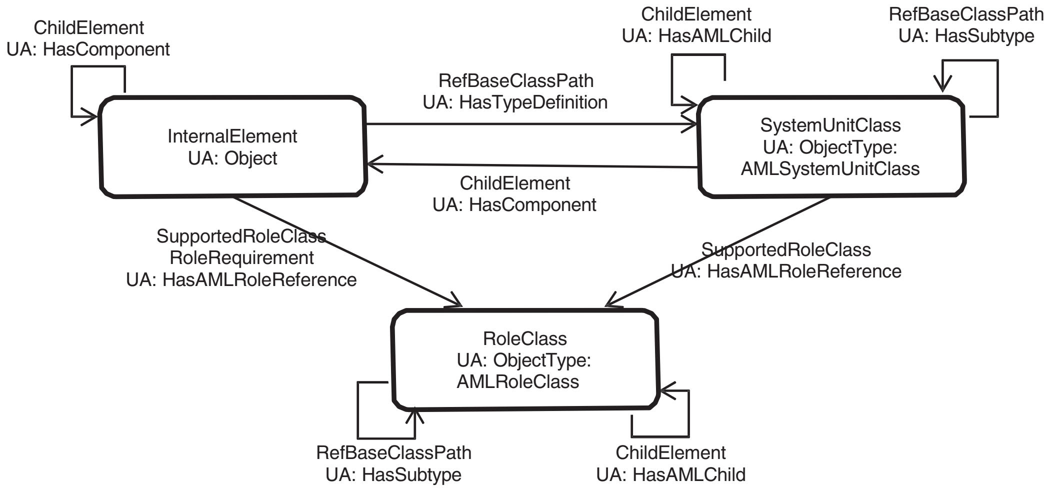

Fig. 8. AutomationML elements in OPC UA.

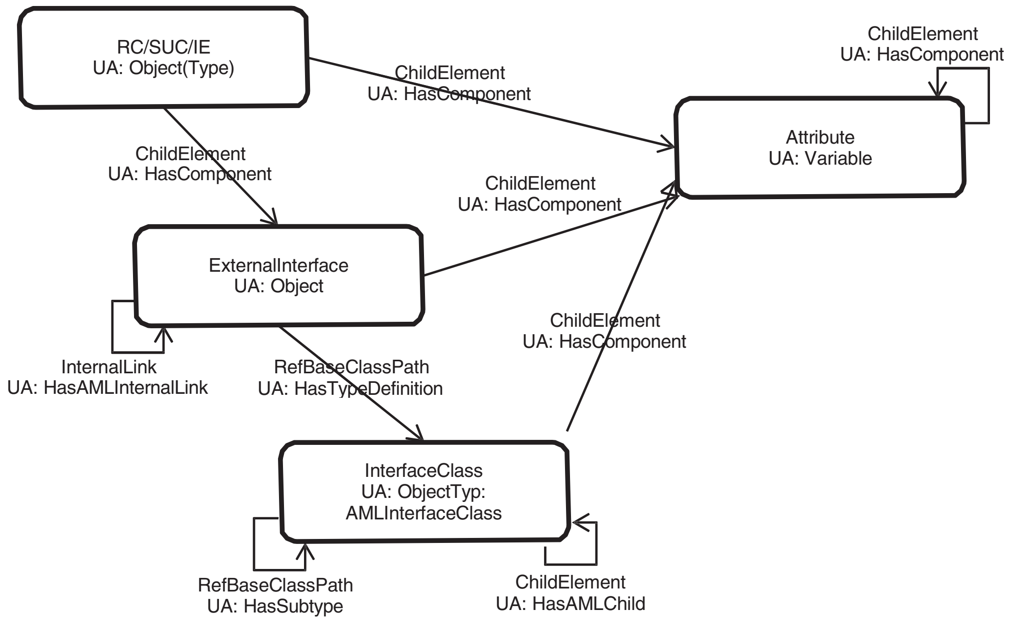

Fig. 9. AutomationML object details in OPC UA.

As already explained the ChildElement relations within class structures will be mapped to the new OPC UA 'HasAMLChild' relations. Child InternalElements will be referenced via 'HasComponent' relations. The inheritance relations within SystemUnitClasses and RoleClasses can be mapped to OPC UA 'HasSubtype' relations.

The mappings of the AutomationML object details (Figure 5) are depicted in Figure 9. The analogy between the figures is obvious.

There is one new OPC UA type used:

- ReferenceType 'HasAMLInternalLink'
  - Subtype of 'NonHierarchicalReferences'
  - Symmetric reference
  - SourceNode and TargetNode should be an ExternalInterface which must not located in RoleClasses.

The Attributes and ExternalInterfaces will be referenced via 'HasComponent' and not via 'HasAMLChild' relations. The 'HasAMLChild' relation is only used within class/type structures.

The InterfaceClass is the type definition of the ExternalInterface therefore it will be assigned via 'HasTypeDefinition'. Similarly to the RoleClasses and SystemUnitClasses, the relations between InterfaceClasses are mapped to 'HasAMLChild' within the XML tree structure and to 'HasSubtype' concerning inheritance relation.

4.3. Example model transformed from AutomationML into OPC UA information model

The example shown in Figure 6 will now be redefined within OPC UA. The next two figures show the result of a manual mapping. The OPC UA modelling syntax shown in Figure 1 and Figure 2 is used.

Figure 10 depicts the class structures of the example. On the right side of the figure you can see the inheritance structure of the roles. On the top the role 'SpecialRobot' downwards past the roles 'Robot', 'Resource' and 'AutomationMLBaseRole' to the basic types 'CAEXObjectType' (AutomationML) and 'BaseObjectType' (OPC UA). The role 'Robot' has two components, an interface 'CommunicationInterface' and an attribute 'axes'.

On the top left of Figure 10 there are the SystemUnitClasses. On top the 'ABCRobot', with reference to the role 'Robot', has the same components as the role. There is also the SystemUnitClass 'ABCLine' and the common parent type 'AutomationMLBaseSytemUnit', which is the father of all SystemUnitClasses in OPC UA models.

In the middle you can find the InterfaceClass 'Communication' derived by the 'AutomationMLBaseInterface'. Every interface (see 'Robot' and 'ABCRobot') has this InterfaceClass as type definition.

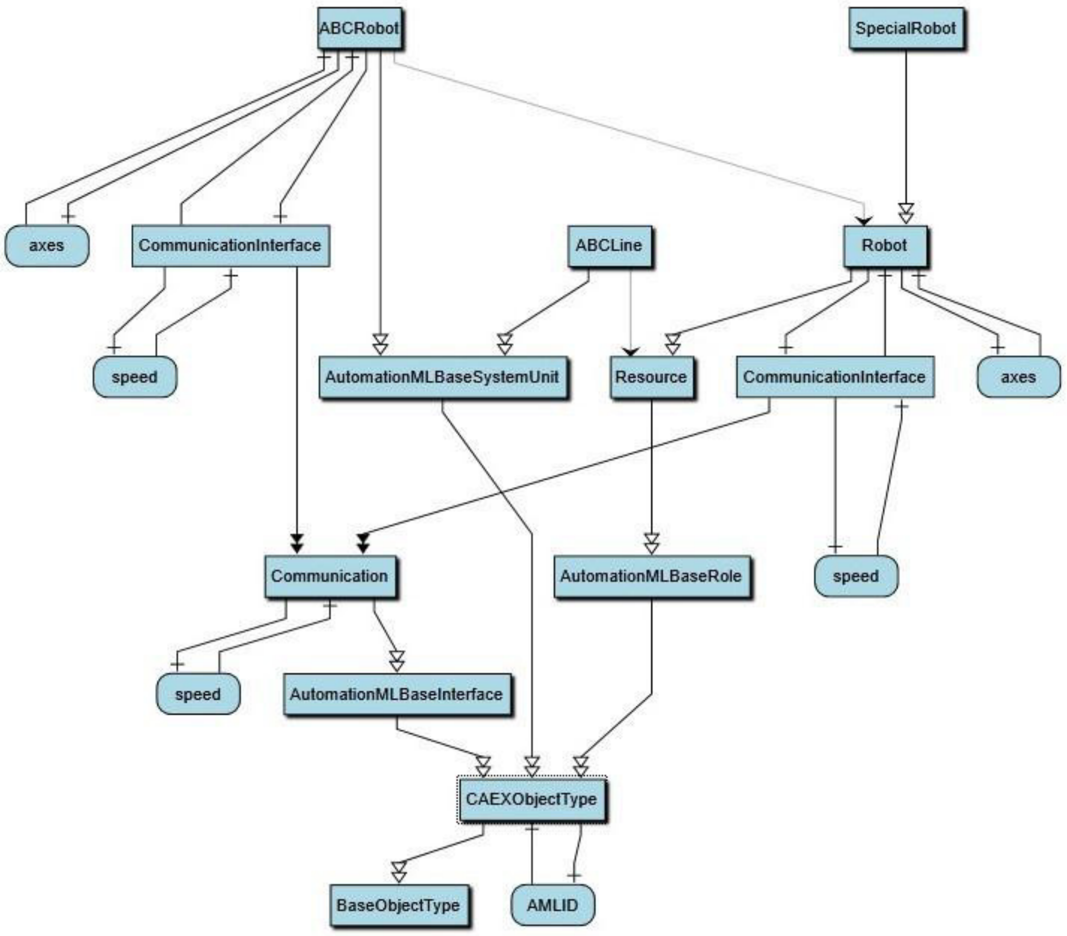

Fig. 10. Example class tree in OPC UA.

The mapping of the InstanceHierarchy is shown in Figure 11. The object on the top 'AMLLibraries' from type 'AMLLIbrariesFolderType' is the folder for all AutomationML libraries. In this case, it contains also the InstanceHierarchy 'TestProject'. This InstanceHierarchy is shown at the left and contains the 'MainLine' and the two robots. Every instance object has its SystemUnitClass type reference. The 'RobotII' has additionally a reference to the role 'SpecialRobot'. All libraries and their classes are shin, certainly without the defining components.

5. Summary and outlook

Goal of this contribution was to explain how to combine the two standards OPC UA and AutomationML with the goal to simplify the creation of OPC UA information models. Data exchange formats like AutomationML can be operationalized with specific OPC UA information models. There is a benefit from the analogies between the two standards, such as the type-instance concept, the differentiation between objects and attributes and typecast of interfaces and relations.

The basic types of AutomationML can be defined in a companion specification for OPC UA. In this way, basic structures like RoleClasses and SystemUnitClasses can be predefined in OPC UA. This could lead into a standardised guideline to combine AutomationML with OPC UA. The publication of this possible companion specification can reduce errors during the implementation and increase reusability. If mapping rules are well defined the transformation can run automatically.

Currently it is a big effort to create an information model for an OPC UA server. Reusing of information already modelled in AutomationML can simplify the usage of OPC UA.

After a short introduction this contribution described the modelling concepts of OPC UA and the plant description format AutomationML. Analogies and differences between the formats are discussed and an AutomationML specific extension of the OPC UA basic types was proposed. Finally a short AutomationML example was given which was then mapped to OPC UA.

First implementation and evaluation of the concepts will be realized in the European project SkillPro (Skill-based Propagation of "Plug&Produce"-Devices in Reconfigurable Production Systems by AML) [13]. An OPC UA server will be developed as part of the MES which will receive asset descriptions in the AutomationML format.

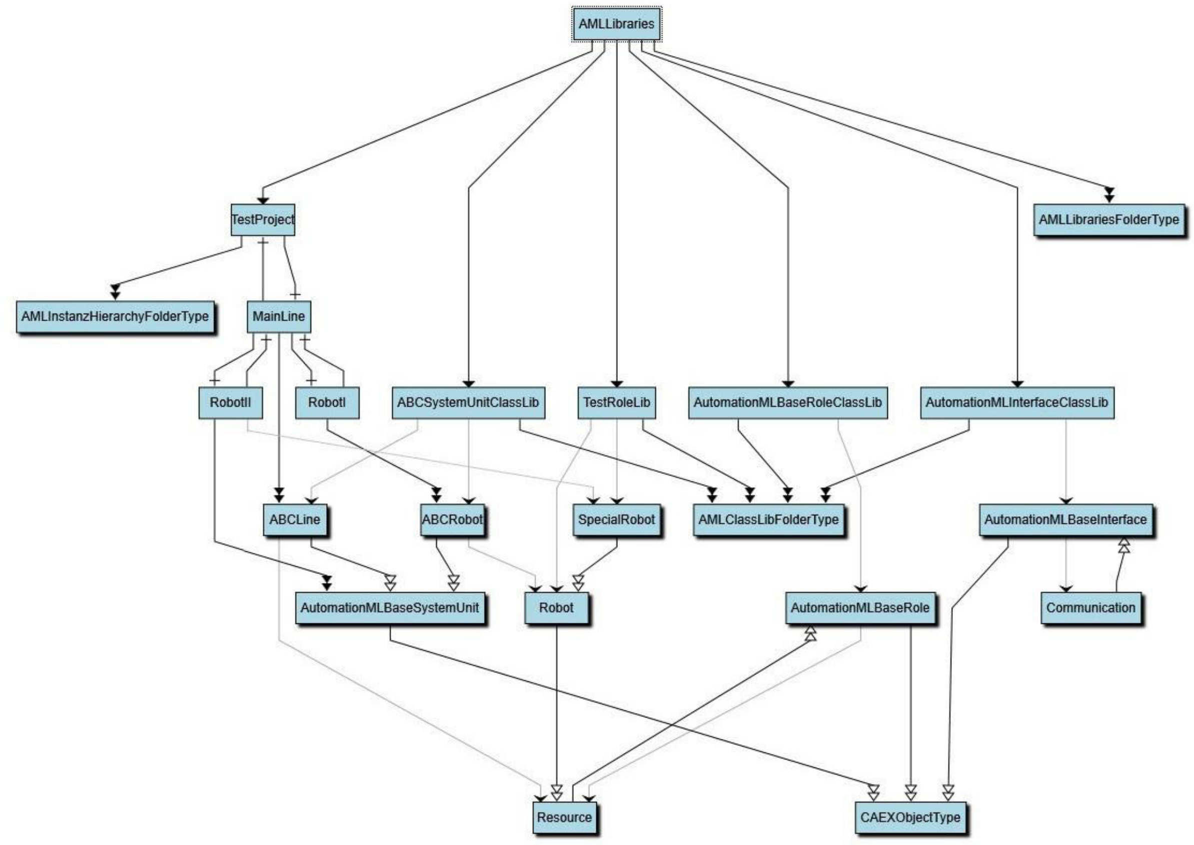

Fig. 11. Example InstanceHierarchy in OPC UA.

References

[1] IEC 62541 (all parts):2010-2012 OPC Unified Architecture.
[2] Mahnke, W., Leitner, S-H. Damm, M. (2009): OPC Unified Architecture: Springer.
[3] IEC 62714 (all parts), Engineering data exchange format for use in industrial systems engineering – Automation Markup Language AML (whitepaper available at http://www.automationml.org/).
[4] Draht, R. (Hrsg.) (2010): Datenaustausch in der Anlagenplanung mit AutomationML: Springer
[5] Sauer, O., Jasperneite, J.: Adaptive information technology in manufacturing. Proceedings of the CIRP International Conference on Manufacturing Systems, Madison/Wisconsin, 01.-03.06.2011.
[6] Schleipen, M., Sauer, O., Wang, J.: Semantic integration by means of a graphical OPC Unified Architecture (OPC UA) information model designer for Manufacturing Execution Systems. In: Sihn/Kuhlang (Ed.), Sustainable Production and Logistics in Global Networks. 43rd CIRP International Conference on Manufacturing Systems, 26-28 May 2010, Vienna: Neuer Wissenschaftlicher Verlag, pp. 633-640.
[7] IEC 62541-1:2010 OPC Unified Architecture Part 1: Overview and Concepts
[8] IEC 62541-3:2010 OPC Unified Architecture Part 3: Address Space Model
[9] OPC Foundation, August 21, 2013 (available at http://opcfoundation.org/Default.aspx/02_news/02_news_display.asp?id=1064&MID=News)
[10] IEC 62424:2008, Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools.
[11] COLLADA 1.4.1: March 2008 COLLADA – Digital Asset Schema Release 1.4.1 (available at http://www.khronos.org/files/collada_spec_1_4.pdf).
[12] PLCopen XML 2.0:December 3rd 2008 and PLCopen XML 2.0.1: May 8th 2009, XML formats for *IEC 61131*-3 (available at <http://www.plcopen.org>*/)*
[13] SkillPro - Skill-based Propagation of "Plug&Produce"-Devices in Reconfigurable Production Systems by AML, EU FP7 (available at www.skilpro-project.eu), November 01, 2013.