



<AutomationML/>

**The Glue for Seamless
Automation Engineering**

**Application Recommendations:
Automation Project Configuration**

Document Identifier: AR APC, V 1.0.0

State: April 2017

©AutomationML consortium

Version 1.0.0, April 2016

Contact: www.automationml.org

Table of contents

Table of contents	3
List of figures	5
List of tables	6
1 Introduction.....	7
1.1 Basics.....	7
1.2 Scope	7
1.3 References.....	8
2 General notes regarding exchange of Automation Project Configuration data	9
2.1 Data exchange workflow.....	9
2.2 Possibilities of configuration.....	10
2.3 Recommended workflow.....	10
2.3.1 Providing an initial PLC project as basis for electrical engineering	10
2.3.2 ECAD engineering.....	10
2.3.3 PLC engineering.....	13
3 Automation Project Configuration data structures in AutomationML	14
3.1 Basic concept.....	14
3.1.1 Export from ECAD to AutomationML	14
3.1.2 Import from AutomationML into PLC.....	15
3.2 The neutral model: Automation Project Configuration data	15
3.2.1 Basic ideas	16
3.2.2 Contents of data exchange	17
3.2.3 Automation Project Configuration data exchange data model.....	17
4 Guideline for the use of the ECAD model in practical applications	23
5 Modelling of Automation Project Configuration data with AutomationML	24
5.1 RoleClassLibrary.....	24
5.1.1 AutomationProject	26
5.1.2 DeviceUserFolder.....	27
5.1.3 Subnet	27
5.1.4 Device	28
5.1.5 DeviceItem	29
5.1.6 TagTable	30
5.1.7 TagUserFolder	31
5.1.8 Node.....	31
5.1.9 CommunicationInterface	32
5.1.10 IoSystem	32
5.1.11 CommunicationPort.....	32
5.2 InterfaceClassLibrary	33
5.2.1 Tag	34
5.2.2 Channel	35
5.2.3 Naming and Escaping	35
6 Practical examples	36
6.1 Running example	36
6.2 Additionally applied role classes.....	37

6.3	Additionally applied interface classes	37
6.4	SystemUnitClassLibrary	37
6.4.1	Network card	39
6.4.2	Digital input module	39
6.4.3	Digital output module	40
6.4.4	Programmable fieldbus controller	41
6.4.5	Modular fieldbus I/O system	42
6.4.6	Sensor	43
6.4.7	Drive	43
6.4.8	Wire	43
6.4.9	Network	44
6.4.10	Master slave communication network	44
6.4.11	PC	44
6.5	InstanceHierarchy	45

List of figures

Figure 1 – Automation Project Configuration between ECAD and PLC tool	9
Figure 2 – Data exchange workflow	9
Figure 3 – Example for PLC configuration and graphical placement	10
Figure 4 – Example for stations and bus data configuration	11
Figure 5 – Example for symbolic address configuration	11
Figure 6 – Example for error check	12
Figure 7 – Example for export from an ECAD tool	12
Figure 8 – Example for import into a PLC	13
Figure 9 – Example for result of import into a PLC	13
Figure 10 – Basic concept for ECAD-PLC data exchange	14
Figure 11 – Basic concept for ECAD-export (EPLAN example)	14
Figure 12 – Basic concept for PLC-import (S7 example)	15
Figure 13 – Coupling CAE and PLC	15
Figure 14 – Objects and parameters of the Automation Project Configuration data exchange	18
Figure 15 – Procedure for use of ECAD in AutomationML	23
Figure 16 – AutomationProjectConfigurationRoleClassLib in AutomationML Editor view	24
Figure 17 – AutomationProjectConfigurationRoleClassLib as XML representation	26
Figure 18 – AutomationProjectConfigurationInterfaceClassLib in AutomationML Editor view	33
Figure 19 – AutomationProjectConfigurationInterfaceClassLib as XML representation	33
Figure 20 – Example system	36
Figure 21 – Example system schematic representation	36
Figure 22 – CommunicationRoleClassLib	37
Figure 23 – CommunicationInterfaceClassLib	37
Figure 24 – SystemUnitClassLibrary of the example	38
Figure 25 – Network card model	39
Figure 26 – Digital input module model	39
Figure 27 – Digital output module model	40
Figure 28 – Programmable fieldbus coupler model	41
Figure 29 – Modular controller model	42
Figure 30 – Sensor model	43
Figure 31 – Drive model	43
Figure 32 – Wire model	43
Figure 33 – Network model	44
Figure 34 – Master slave communication network model	44
Figure 35 – PC model	44
Figure 36 – Upper layer hierarchy elements	45
Figure 37 – Modelling elements of turntable	45
Figure 38 – Modelling elements of control cabinet	46
Figure 39 – Modelling of communication wiring based on wire elements and internal links	47
Figure 40 – Modelling elements of wiring	48
Figure 41 – Modelling of physical wiring based on wire objects and internal links	48
Figure 42 – Modelling of device assignment to networks	49
Figure 43 – WagoModularController instance with relevant parameters described by attributes	50

List of tables

Table 1 – Overview of AutomationML parts.....	7
Table 2 – Definition AutomationProject	26
Table 3 – Definition DeviceUserFolder	27
Table 4 – Definition Subnet.....	27
Table 5 – Definition Device	28
Table 6 – Definition DeviceItem	29
Table 7 – Definition TagTable	30
Table 8 – Definition TagUserFolder	31
Table 9 – Definition Node	31
Table 10 – Definition CommunicationInterface.....	32
Table 11 – Definition IoSystem	32
Table 12 – Definition CommunicationPort	32
Table 13 – Definition Tag.....	34
Table 14 – Definition CommunicationPortInterface	34
Table 15 – Definition Channel.....	35

1 Introduction

A very frequently occurring task within the planning process of production and automation systems is the exchange of automation project configuration information of automation system devices between ECAD and PLC systems. To avoid multiple engineering in the participating systems ECAD and PLC systems need an interface for sharing this information.

In case of beginning engineering in the ECAD tool certain rules must be observed to get the hardware information in the correct location in the PLC tool. In case of beginning engineering in the PLC tool non placed functions must be placed and operated in the ECAD tool.

This application recommendation describes these workflows and the method of hardware configuration modelling using AutomationML.

1.1 Basics

The data exchange format AutomationML which is standardising in the IEC 62714 standard is a neutral, free, and XML-based data format. It has been developed in order to support the data exchange between engineering tools in a heterogeneous engineering tool landscape.

Due to the different aspects of AutomationML the IEC 62714 consists of different parts.

Table 1 – Overview of AutomationML parts

Part	Title	Description
Part 1	Architecture and general requirements	This part specifies the general AutomationML architecture, the modelling of the engineering data, classes, instances, relations, references, hierarchies, basic AutomationML libraries and extended AutomationML concepts.
Part 2	Role class libraries	This part specifies additional AutomationML libraries.
Part 3	Geometry and kinematics	This part specifies the modelling of geometry and kinematics information.
Part 4	Logic	This part specifies the modelling of logics, sequencing, behaviour and control related information.
Whitepaper	Communication	This Whitepaper describes the modelling of Communication mechanisms in AutomationML
Whitepaper	AutomationML and eCI@ss integration	This Whitepaper describes the integration of eCI@ss in AutomationML

Further parts may be added in the future in order to e.g. interconnect further data standards to AutomationML.

1.2 Scope

This application recommendation proposes a modelling method of automation project configuration data by means of the engineering data format AutomationML. It will describe the recommended use of role and interface classes as well as the recommended structures to be considered within the instance hierarchy of an AutomationML project.

1.3 References

The following documents are referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Extensible Markup Language (XML) 1.0:2004, W3C Recommendation (available at <<http://www.w3.org/TR/2004/REC-xml-20040204/>>)

IEC 62424:2008, Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools

Whitepaper AutomationML Part 1 – AutomationML Architecture, October 2014

Whitepaper AutomationML Part 2 –AutomationML Role Libraries, October 2014

Whitepaper AutomationML Part 4 –AutomationML Logic, May 2010

Whitepaper AutomationML– AutomationML Communication, September 2014

Whitepaper AutomationML– AutomationML and eCI@ss Integration, November 2015

2 General notes regarding exchange of Automation Project Configuration data

ECAD tools and PLC tools have different views of automation system information. Whereas ECAD tools depict all electrical detail information of devices applied within automation systems in PLC tools only a logical compilation of the automation devices is used. So in ECAD tools there are defined e.g. devices which are involved in an automation system, voltage connectors which are used for power supply of the devices, and wire types which are used to connect devices. But these are not used in PLC tools. On the other side in PLC tools there are device and control application specific conditions defined e.g. baud rates which are used within the communication connections, control code variables which are associated to control device inputs and outputs, and control application codes. But these are not needed in ECAD tools. Nevertheless, both types of tools have some information in common. For example, the wiring of a certain automation device to a PLC defines the address the device can be accessed within the PLC. This must be considered by development of import and export tools. Figure 1 shows the scope of this application recommendation.

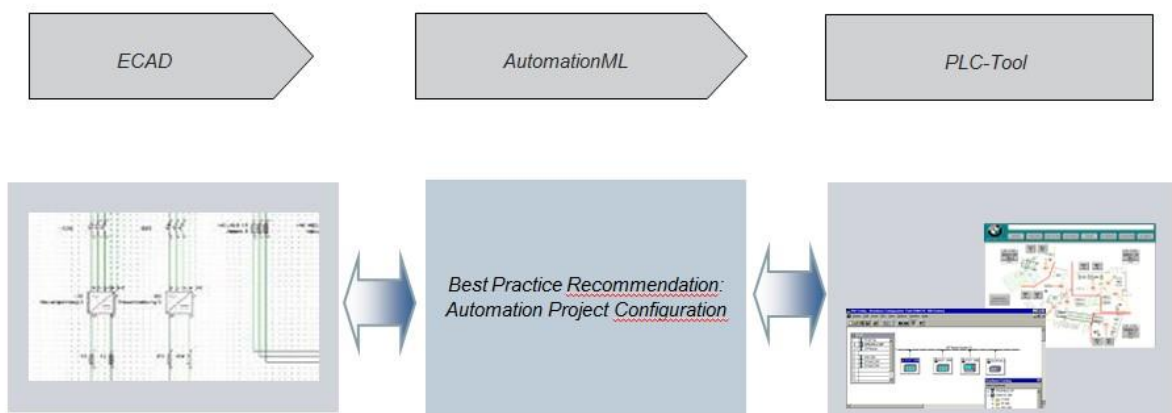


Figure 1 – Automation Project Configuration between ECAD and PLC tool

Beyond the named engineering tools for ECAD and PLC programming also other tools can be interested in the common data set of both tools. For example tools for mechanical engineering (MCAD) can be interested in the devices to be wired and documentation tools can be interested in the wiring structure reached. Nevertheless, within this document only ECAD and PLC programming tools are considered knowing that more engineering tools can benefit from importing the modelled information.

2.1 Data exchange workflow

Usually, in a production system engineering process the construction phase in the PLC project will begin later than in the ECAD system because the completion of the ECAD documents is the base for the production of the control cabinet. The combination with the software within the plant and the following commissioning will not take place before all control cabinets are completed.

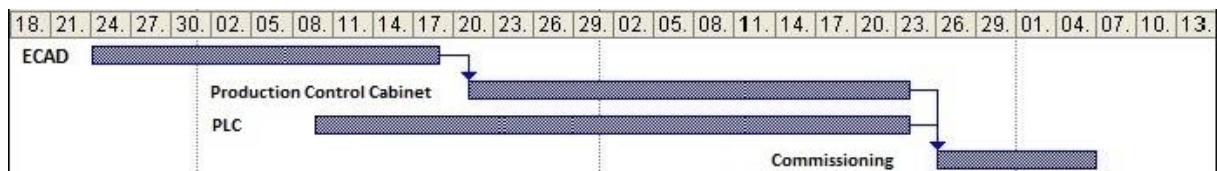


Figure 2 – Data exchange workflow

So the PLC engineer will usually attend later to the project than the ECAD engineer. Nevertheless at an early point of time (during ECAD engineering) the automation project configuration of the plant must be defined because the ECAD documents must be generated and the parts must be ordered.

2.2 Possibilities of configuration

ECAD systems normally can handle the components of different PLC manufacturers which have certain analogies from a point of view of electrical hardware. But additionally there are system specific / manufacturer specific parameters. Therefore only the engineering system of the PLC manufacturer can guarantee a complete and comfortable handling of all parameters of a hardware component. So the configuration of the PLC system should be done as far as possible within the engineering system of the PLC manufacturer.

2.3 Recommended workflow

Accordingly to the described criteria in most cases the following workflow is established.

- Engineering the basic device configuration within the PLC project of the PLC programming tool and exporting it to ECAD tool
- Importing PLC project to ECAD tool, engineering of the ECAD project, and exporting the ECAD project to PLC programming tool
- Importing ECAD project into PLC programming tool and engineering of the PLC project

2.3.1 Providing an initial PLC project as basis for electrical engineering

If no ECAD project exists so far, the ECAD engineer first of all defines a raw project within the engineering system of the PLC manufacturer, the PLC programming tool. The ECAD engineer selects all needed components and defines the bus topology in close cooperation with the PLC engineer who has to implement the requested functions later on. This close cooperation ensures a high consistency regarding the selected hardware components. The automation project configuration will be exported from the engineering system of the PLC manufacturer and imported into the ECAD tool.

2.3.2 ECAD engineering

Based on the existing ECAD project the ECAD engineer executes the complete hardware construction, sometimes with slight adaptations. During this process the symbolic names for variables, tags or signals can be defined too. So the PLC configuration is done under the following conditions:

- PLC configuration can be imported from PLC programming system
- Configuration via graphical placement on overview page or navigator
- PLC-device selection carried out from ECAD database
- Drag&Drop on pages from navigator

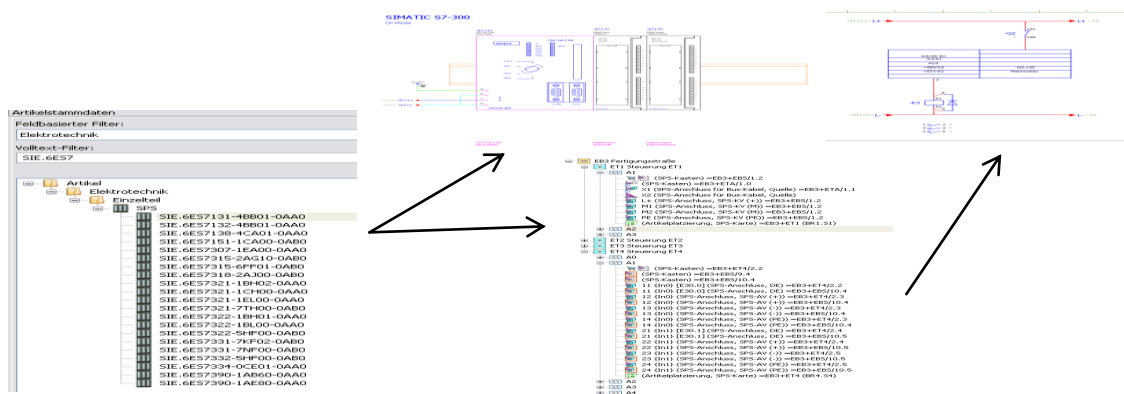


Figure 3 – Example for PLC configuration and graphical placement

When the PLC configuration is completed the stations and bus-data are engineered. This is normally done by manual assignment of cards to CPUs and devices to bus via device properties.



Figure 4 – Example for stations and bus data configuration

In the next step the tag list is to be engineered, i.e. a list of tags, variables or signals (symbolic address) of hardware related tags:

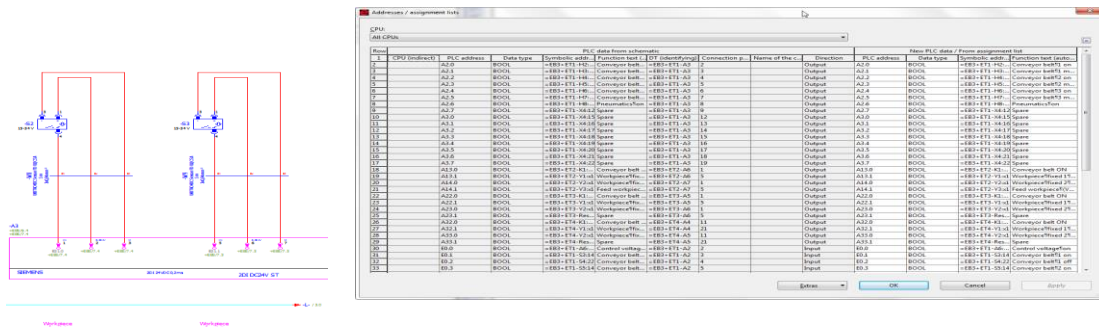


Figure 5 – Example for symbolic address configuration

Finally a simple error check should be performed, e.g.:

- Double usage of I/O / bus or symbolic addresses
- Missing addresses.
- Simple rules, e.g. slot 3 reserved for IM-module

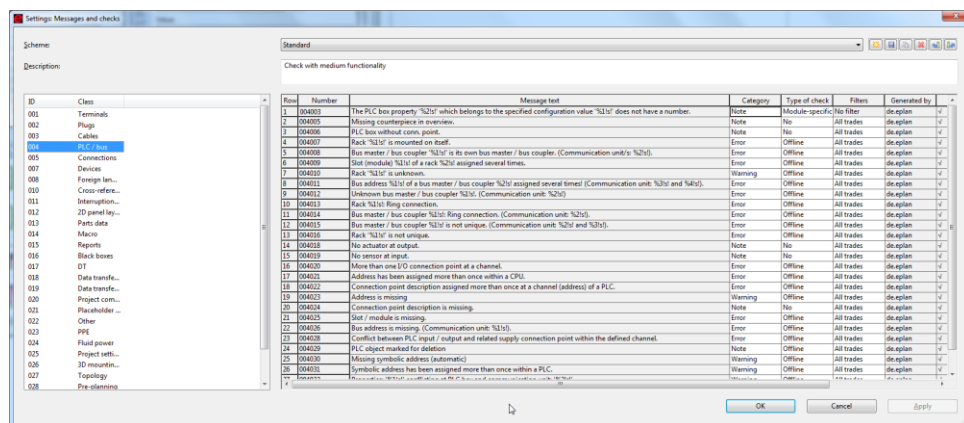


Figure 6 – Example for error check

Now the ECAD project can be exported to AutomationML dependent from the implementation in the ECAD tool and imported to the PLC programming tool.

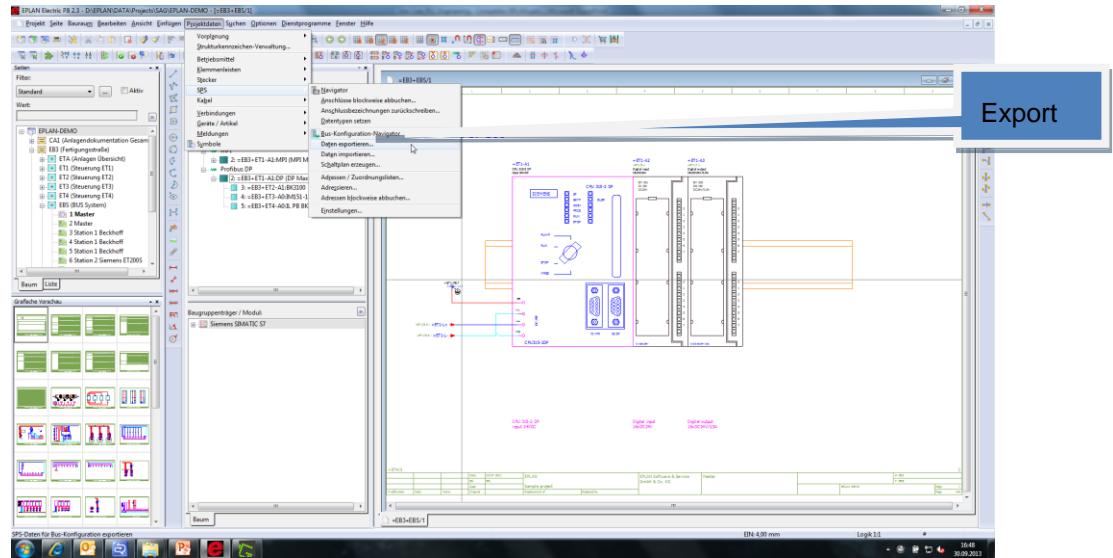


Figure 7 – Example for export from an ECAD tool

2.3.3 PLC engineering

At a later point in time the PLC programmer will begin the engineering based on the already developed ECAD project. So at this point of the engineering process the export function of the ECAD system and the import function of the PLC system should be used to check possible changes and verify the equality of both configurations.

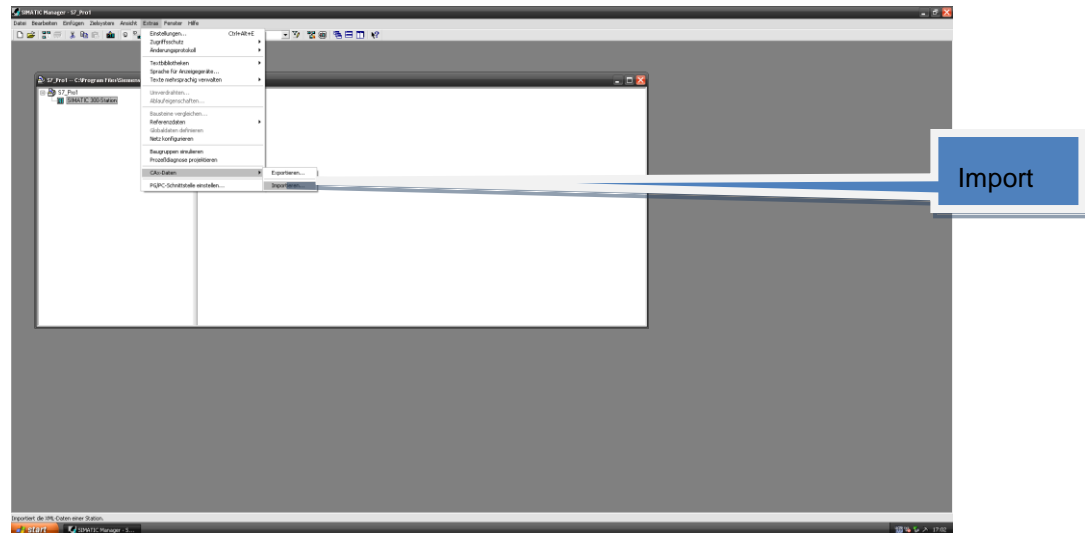


Figure 8 – Example for import into a PLC

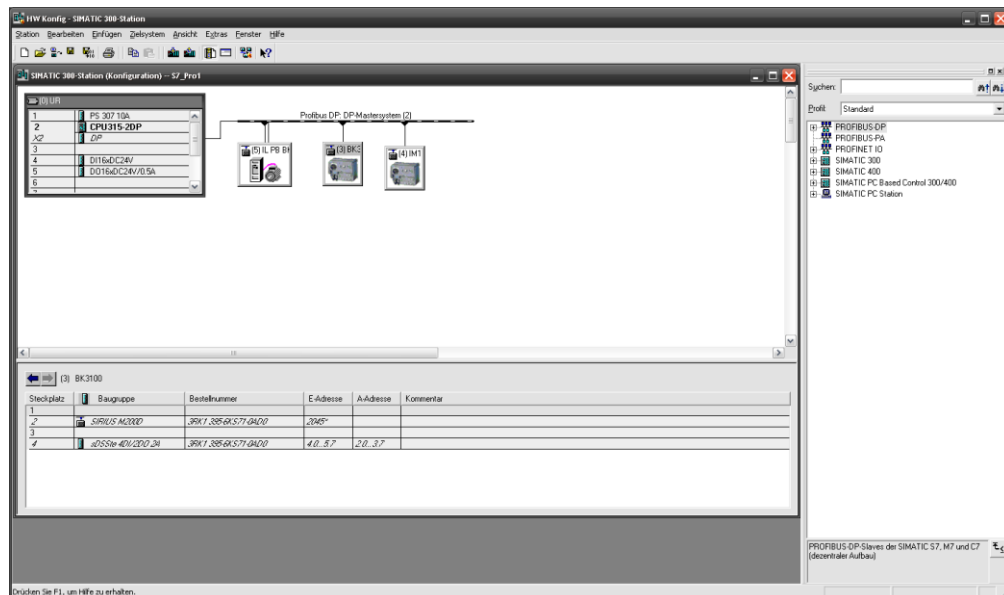


Figure 9 – Example for result of import into a PLC

3 Automation Project Configuration data structures in AutomationML

In the following chapter a concept is defined how Automation Project Configuration data can be represented in AutomationML.

3.1 Basic concept

For using AutomationML as a neutral exchange format for Automation Project Configuration data the PLC-specific interfaces of the different PLC manufacturers must be decoupled. This guarantees an independence of the further development of PLC-tools as well as of the further development of ECAD tools. Furthermore the transformation and implementation should be as easy as possible for ECAD and PLC-vendors likewise. Therefore already existing models should be used as far as possible.

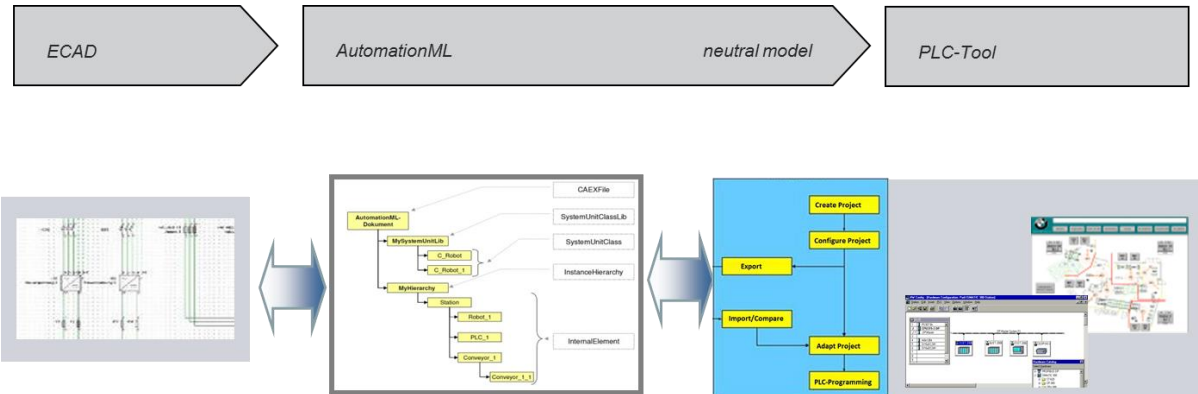


Figure 10 – Basic concept for ECAD-PLC data exchange

Using a neutral model allows

- definition of PLC-Tool independent roles in AutomationML
- definition of PLC-specific SystemUnitClasses for different ECAD- and PLC-Tools / vendors in AutomationML
- definition of PLC-specific InterfaceClasses in AutomationML

3.1.1 Export from ECAD to AutomationML

The following figure shows the detailed export of Automation Project Configuration data based on an EPLAN example:

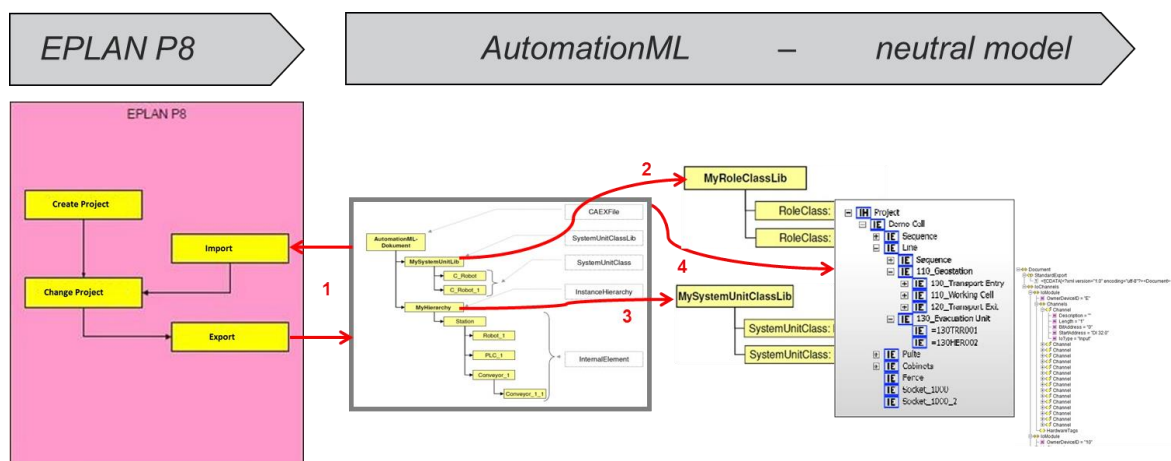


Figure 11 – Basic concept for ECAD-export (EPLAN example)

1. Export / Import of Automation Project Configuration data from / to AutomationML
2. Manufacturer independent roles in AutomationML
3. Neutral SystemUnitClasses in AutomationML
4. Topology in AutomationML (neutral model)

3.1.2 Import from AutomationML into PLC

The following figure shows the detailed import of Automation Project Configuration data based on an S7 example:

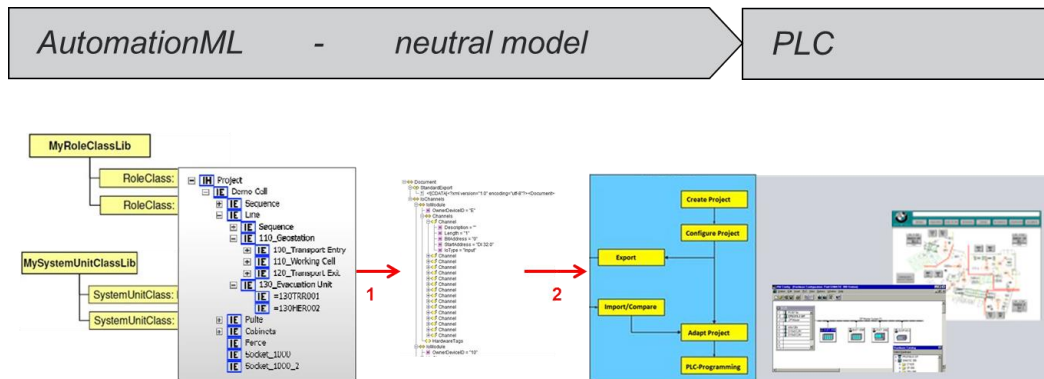


Figure 12 – Basic concept for PLC-import (S7 example)

1. Import of ECAD data from AutomationML (neutral model)
2. Import from neutral model into manufacturer specific PLC tool (Example S7)

3.2 The neutral model: Automation Project Configuration data

The aim is to support the engineering workflow between ECAD systems and PLC engineering systems. Providing standardized interfaces for the data exchange between PLC and ECAD systems are mandatory. The interfacing to ECAD systems has as further target group all ECAD manufacturers:

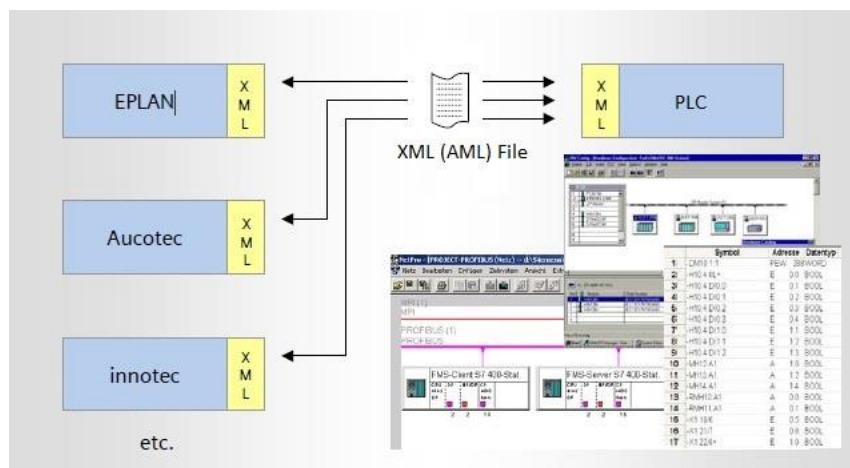


Figure 13 – Coupling CAE and PLC

So we have to consider the different ECAD systems and CAE manufacturer. Therefore the term “ECAD” stands for the different CAE, E-CAD and E-CAE formats depending on the different existing ECAD systems. Based on this the implementation of a neutral “proxy ECAD”-format in AutomationML shall be defined in the following chapter. Furthermore the already existing concepts of the leading ECAD-Tool manufacturer and PLC manufacturer shall be considered to ensure an “as easy as possible” implementation of this neutral model for all ECAD and PLC manufacturer.

3.2.1 Basic ideas

The Automation Project Configuration data can be modelled by using AutomationML. Therefore the modelling methodology is based on the concepts of AutomationML topology modelling using CAEX defined in Part 1 of the AutomationML standard. Additional provisions are added to the basic definitions to fulfil the special requirements that arise from data exchange with ECAD tools. The Automation Project Configuration data modelling methodology enables the development of a self-containing model. No dependencies to other models are mandatory.

For modelling of Automation Project Configuration data a vendor neutral Automation Project Configuration data structure will be defined. It represents in its base structure a neutral object model of PLC systems.

The data exchange is based on complete information about the objects. This means that the Automation Project Configuration data always holds the complete data of the object itself and not only delta information.

The export/import granularity is at the level of hardware stations as PLC engineering systems always operate at a station level. The export/import will either support exchange of data referring to one or more stations (e.g. a complete project) or only parts of a station (e.g. one single module). The requirement here always is that the “environment“, the part lives in, is also part of the data exchange.

Only PLC hardware configuration information of automation devices including some relevant parameter and symbols/tags related to the hardware objects are in scope of this data exchange. Additionally information about the networks these hardware configurations are connected to is part of this exchange format.

Only a subset of all data provided by PLC hardware configuration is relevant for data exchange with ECAD systems. Due to the electrical view of the plant handled by ECAD tools, these tools can only deliver a very general subset of information belonging to the PLC hardware objects. Specific parameter settings are the domain of PLC specific hardware-configuration tools and the specific object managers. They only can be handled in the manufacturer specific tool.

Besides the standard devices in the PLC hardware catalogues, there are some types of device items that need additional descriptions. Examples for this are GSD or GSDML descriptions. Devices and device items like norm slaves or GSDML based IODevices can only be instantiated in a PLC configuration if the appropriate description / package is installed. It is the responsibility of the PLC programmer to make sure that the correct and up to date device item information (GSD, etc.) is available. But the AutomationML based ECAD data exchange file can provide information about the needed device item information.

Some of the ECAD systems are capable to provide that information about a needed description file. Others also can provide the file itself. Therefore it is allowed to transmit this additional data from the ECAD system to PLC engineering system. Files are expected to be delivered e.g. as a zip file and are unpacked into the same directory as the import file. Thus it should be possible to specify and reference the file in the data exchange file. The user of a PLC system, who is importing the file, can expect to find some or all needed descriptions in the same directory as the import-file. The reference to the description file is inserted as separate properties for the module. It is an anchor to the item description.

3.2.2 Contents of data exchange

An analysis of the already existing proprietary XML-based data exchange files of the leading PLC and ECAD manufacturers regarding the Automation Project Configuration data showed that all data to be exchanged can be grouped in three major categories:

1. HW Data:

These are data concerning parts or devices like a central rack, a slave or a switch. Therefore mostly the term “device” is used for this group of parts. Within these devices there are sub devices or device items like racks, CPUs, power supply, I/O Modules, submodules. Therefore mostly the term “device item” or “subdevice” is used for this group of parts. Additional device items like routers, switches, hubs, repeaters will be supported by the export format. Devices are often grouped in a hierarchical “folder” structure.

2. Symbols / Tags:

Exported and imported are “symbols” and “tags” assigned to a device item. Only hardware oriented symbols/tags are considered here. The symbols/tags are exported with the controller target device item (i.e. the CPU) and not with other device items they might refer to (e.g. an I/O module). Like devices also the tags are often grouped in “tag tables” and in a hierarchical “folder” structure.

3. Networks:

Networks are modelled right below the project as global subnet objects. The link between a network and the device items are modelled as a reference to the network object. There is no reference from the network object to the attached device items. The network parameters are stored at the network object. The parameters concerning a network interface of a given device item, attached to a network, are stored in a net node object at that device item. The communication is often regulated using “channels”, “ports” and “interfaces”.

Additionally the “Whitepaper AutomationML Part 5 – AutomationML Communication, September 2014” already defines an XML based methodology for communication system information exchange among engineering tools developed by AutomationML e.V. These methods shall also be considered when modelling Automation Project Configuration data.

3.2.3 Automation Project Configuration data exchange data model

The consideration of all these mentioned and already existing models leads to the following Automation Project Configuration data Exchange diagram:

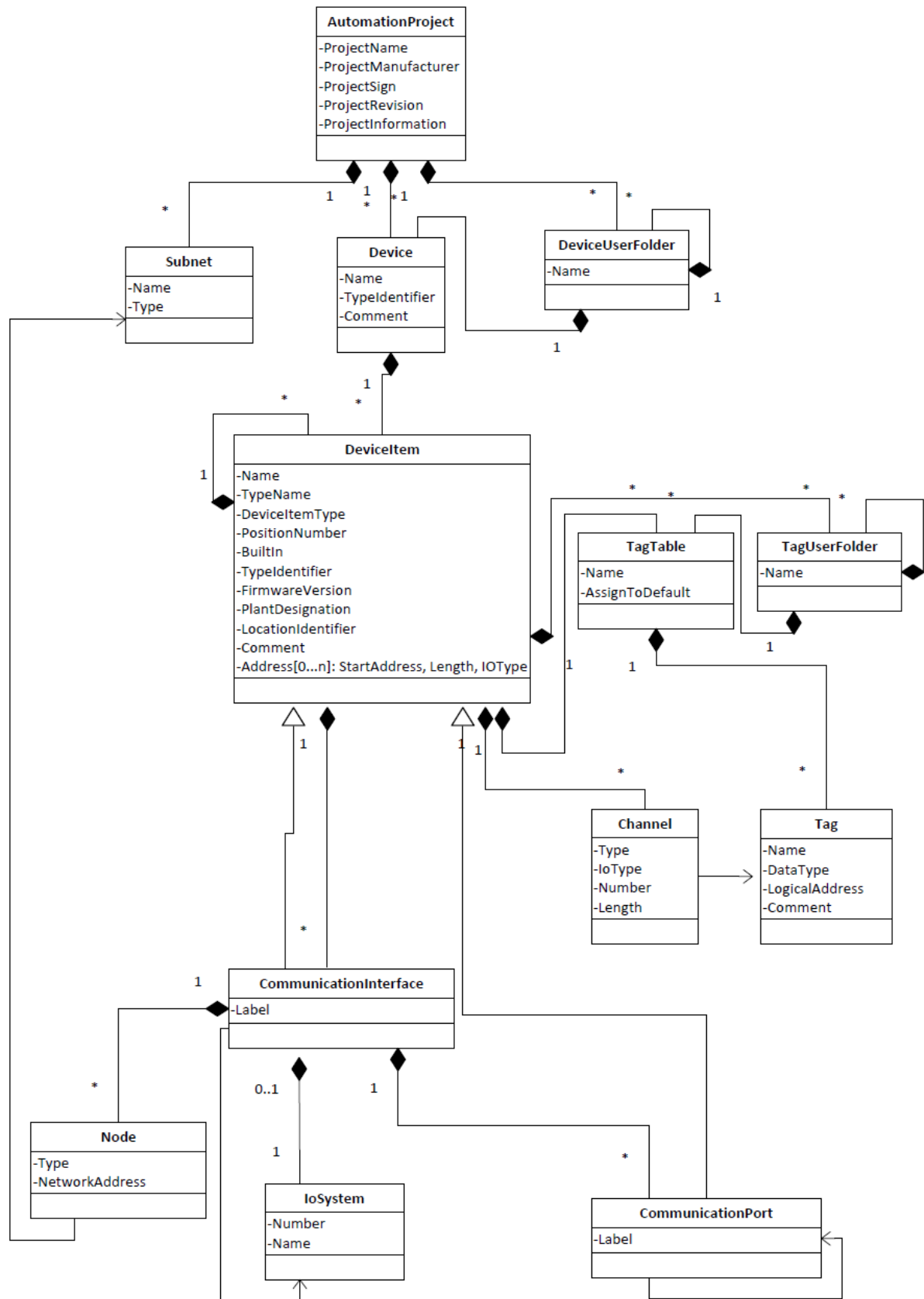


Figure 14 – Objects and parameters of the Automation Project Configuration data exchange

The AutomationML export of Automation Project Configuration data is based on the use of an InstanceHierarchy covering the exported Automation Project Configuration data. The InternalElements of this instance hierarchy will reference appropriate elements in RoleClass Libraries, SystemUnitClass Libraries, and InterfaceClass Libraries.

The objects and parameters shown in the figure above are described as follows. All objects will be modelled as role classes or interface classes derived from classes defined in Whitepaper AutomationML Part 5 and completed with additional attributes already used in the tool landscape of PLC manufacturers. Additional parameters can be defined using eCI@ss integration mechanisms as described in "Whitepaper AutomationML Part 6 – AutomationML and eCI@ss Integration".

3.2.3.1 AutomationProject

An AutomationProject object represents the project from which the export arises. It aggregates all other objects below. The standard parameter for a Project are its "name" (string), the project manufacturer (string), the project sign (string) of the manufacturer, the project revision number (string), and a project information (string) hosting a comment to the project.

3.2.3.2 DeviceUserFolder

A DeviceUserFolder supports the structure of a device within a project. The only and one standard parameter for a DeviceUserFolder is its "name" (string)

3.2.3.3 Subnet

A Subnet object is responsible for storing and managing properties and functionality of networks like Ethernet, PROFIBUS, MPI, etc. A subnet is defined by the physical availability of all subnet participants. All subnet participants have different, unambiguous addresses. The standard parameters of a Subnet object are listed below:

- Name (string):
Name of the Subnet
- Type (string):
The type of the subnet (PROFIBUS, MPI...)

3.2.3.4 Device

A Device object represents a collection in which the individual HW objects of a slave or rack, including the slave or rack HW item, are brought together. Therefore a Device is a Device Item container that serves as a collection of Device Items (in particular hardware items). A Device has to have a unique name within a project. A device can be:

- a central configuration with some racks within (Automation system station with central and extension racks)
- a fix combination of CPU and some I/O modules (e.g. C7),
- a PC station where the PC represents a device,
- a field device,
- a switch

The standard parameters of a Device object are listed below:

- Name (string):
Name of the Device
- TypeIdentifier (string):
Identifier of the device type
- Comment (string):
An optional comment for the device.

3.2.3.5 DeviceItem

A DeviceItem is aggregated by a Device and represents an “abstract” object class for HW modules and submodules (CPU, I/O module, rack, etc.). Whereas a Device represents the logical bracket, the DeviceItems represent more the physical hardware objects.

A DeviceItem can be plugged in another Device Item (e.g. CPU within a rack, sub module within a module). The relative position to the father object is defined by the PositionNumber.

A DeviceItem can also be built in another DeviceItem. These DeviceItems can model a fix combination that cannot be broken up (e.g. C7). The standard parameters of a DeviceItem object are listed below:

- Name (string):
Name of the DeviceItem
- TypeName (string):
Additional type information. Not mandatory but useful for user in case of error.
- DeviceItemType (string):
Classification of the DeviceItem (e.g. CPU)
Additional information. Not mandatory but useful for user in case of error.
- PositionNumber (int):
Slot number where this DeviceItem is plugged in.
- BuiltIn (Boolean):
Flag indicating that this module is a build-in part of another module. This module is automatically created because it is a fixed part of the other module. If omitted this parameter defaults to false.
- TypeIdentifier (string):
Identifier of the device item type
- FirmwareVersion (string):
Specifies the firmware version of e.g. a CPU and might be needed to identify the module correctly (sometimes the order number is not sufficient).
- Comment (string):
An optional comment for the module.
- Address (list attribute):
Address information of device item within device. Most modules have address ranges assigned. There may be e.g. address ranges for input, output channels which are described by their start value and length. So the Address defines the start, length and IO-type. It is modelled as list of address parameters with the sub parameters as listed below:
 - StartAddress (int):
Start of the Address
 - Length (int):
Total width of the module (vendor specific). In the most cases it corresponds to the width of all channels.
 - IoType (string):
Input or Output

Note: In addition to the named standard documents attributes can be added enabling the representation of reference designations following IEC 81346. The following attributes will give two possible examples.

- PlantDesignation IEC (string):
Plant designation for this device item. The PlantDesignation is a product oriented reference designation following IEC 81346.

- **LocationIdentifier IEC (string):**
Location designation for this device item. The LocationIdentifier is a location oriented reference designation following IEC 81346.

3.2.3.6 TagTable

A TagTable supports the structuring of tags. The standard parameters of a TagTable object are listed below:

- **Name (string):**
Name of the TagTable
- **AssignToDefault (Boolean):**
While importing if the TagTable has an attribute 'AssignToDefault' with 'True' value, then all the Tags inside will be imported to an existing default TagTable. In this case the name of the TagTable is ignored by the importing tool. By default, False value is assumed for 'AssignToDefault' attribute if it does not exist while importing.

3.2.3.7 TagUserFolder

The TagUserFolder supports the structuring of TagTables within a DeviceItem. The only and one standard parameter for a TagUserFolder is its "name" (string).

3.2.3.8 Tag

A Tag represents the symbolic name of an I/O date. It provides the logical view on the Channel of a module and is referenced by the associated channel directly.

Tags can only be aggregated by a Tag Table of a CPU. The CPU is represented by a concrete Device Item. The standard parameters of a Tag object are listed below:

- **Name (string):**
Name of the Tag
- **DataType (string):**
Type of the data (e.g. bool, byte word)
- **LogicalAddress (string):**
Logical Address specifies the address of the tag.
- **Comment (string):**
An optional comment specified for the tag

Tags without assigned channels and channels without assigned tags are possible (incomplete engineering)

3.2.3.9 Channel

A Channel is part of an IO module and represents the process interface (e.g. digital or analogue input/output). A channel is part of the DeviceItem which represents the IO module and can only be used in a DeviceItem. The channel refers to tags using a link. The standard parameters of a channel object are listed below:

- **Type (string):**
Analog or Digital
- **IoType (string):**
Input or Output
- **Number (int):**
Number of the channel, starting with 0
- **Length (int):**
Width of the channel (e.g. bit, byte word)

A channel references with a LinkToTag to the associated Tags which are stored at a CPU DeviceItem.

3.2.3.10 CommunicationInterface

A CommunicationInterface is a special type of a DeviceItem acting as a connection point of a device to a network (e.g. network card). The standard parameters of a CommunicationInterface object are listed below:

- Label (string):
Name printed on the item e.g. unique identifier.

3.2.3.11 Node

A Node specifies all the interface related networking information of a network node. (e.g. logical address, subnet mask). A Node belongs to the CommunicationInterface (DeviceItem). The parameters of a node are bus specific characteristics. It is a topology object of a physical connection (cable, glass fibre) between two network stations. Depending on the node type a node can contain different node type specific parameters. The standard parameters of a node object are listed below:

- Type (string):
The Type of the Network (e.g. Ethernet , MPI...).
- NetworkAddress (string):
Network address of this device item. The format depends on the Node type, e.g. a TCP/IP address for an IP network.

Here an example for PROFINET specific attributes:

- DeviceNumber (string):
The device number can be used to identify an IO device (if applicable for the Node type).
- SubnetMask (string):
Subnet mask used by this item (if applicable for the Node type).

3.2.3.12 CommunicationPort

A CommunicationPort is the physical connection to the network (e.g. Ethernet Port). It is a topology object of a physical connection (cable, glass fibre) between two network stations. The standard parameters of a CommunicationPort object are listed below:

- Label (string):
Name printed on the device item e.g. unique identifier.

Ports are aggregated on an Interface which implicitly defines the relationship between the logical (= Interface) and physical (= Port) network connectivity. This aggregation need not be explicitly modelled in AutomationML because it is available via the derivation of Interface and Port from DeviceItem that already defines a generic DeviceItem to DeviceItem aggregation.

3.2.3.13 IoSystem

An IoSystem object is responsible for representing a master – slave relationship typically found in fieldbus systems. Although this relationship depends on a subnet connection between the interfaces, the object model does not enforce this (incomplete engineering). The parent object of the IoSystem object is the interface that acts as the master. All interfaces that act as slaves for this master are linked to the IoSystem object. Please note that the master interface and the slave interfaces are different object instances although they share the same class in the object model. The standard parameters of an IoSystem object are listed below:

- Name (string):
Name of the IOSystem
- Number (int):
Number of the IOSystem.

4 Guideline for the use of the ECAD model in practical applications

To use the previously described method for modelling ECAD in AutomationML four steps are required.

In a first step the ECAD RoleClassLib must be generated or imported i.e. the appropriate RoleClassLib has to be defined. This RoleClassLib contains the derivation of the ECAD Roles from the generic communication model of AutomationML.

Next the Interfaces must be generated or imported, i.e. the appropriate InterfaceClassLib has to be defined. This InterfaceClassLib contains the derivation of the ECAD Roles from the generic communication model of AutomationML.

In step 3 the SystemUnitClasses for the engineering domain must be identified and modelled as templates for further use. Here the structure of the Devices, DeviceItems... can be modelled especially with respect to the relevant properties to be considered. Therefore, appropriate <InternalElement>'s and attributes are added.

Finally, the defined structure can be used to model a practical system in the InstanceHierarchy.

This procedure is depicted in the following figure:

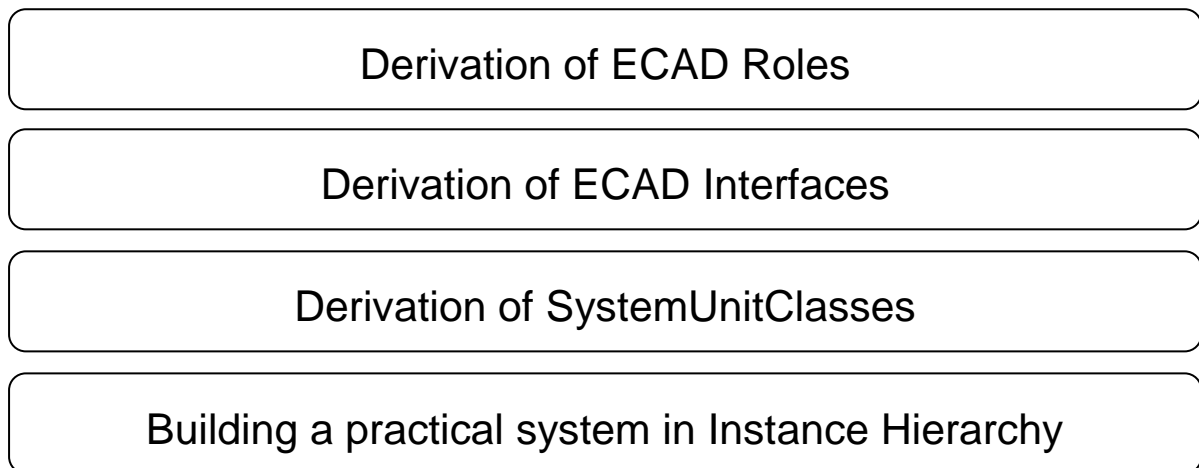


Figure 15 – Procedure for use of ECAD in AutomationML

5 Modelling of Automation Project Configuration data with AutomationML

5.1 RoleClassLibrary

Basement of the modelling are the required role classes. Facing the required model elements there are role classes especially required for Automation Project Configuration data modelling derived from role classes used for communication system modelling defined in AutomationML Whitepaper – Communication or derived from AutomationML basic roles defined in AutomationML Whitepaper – Architecture and general requirements.

The following figures represent the defined role class library.

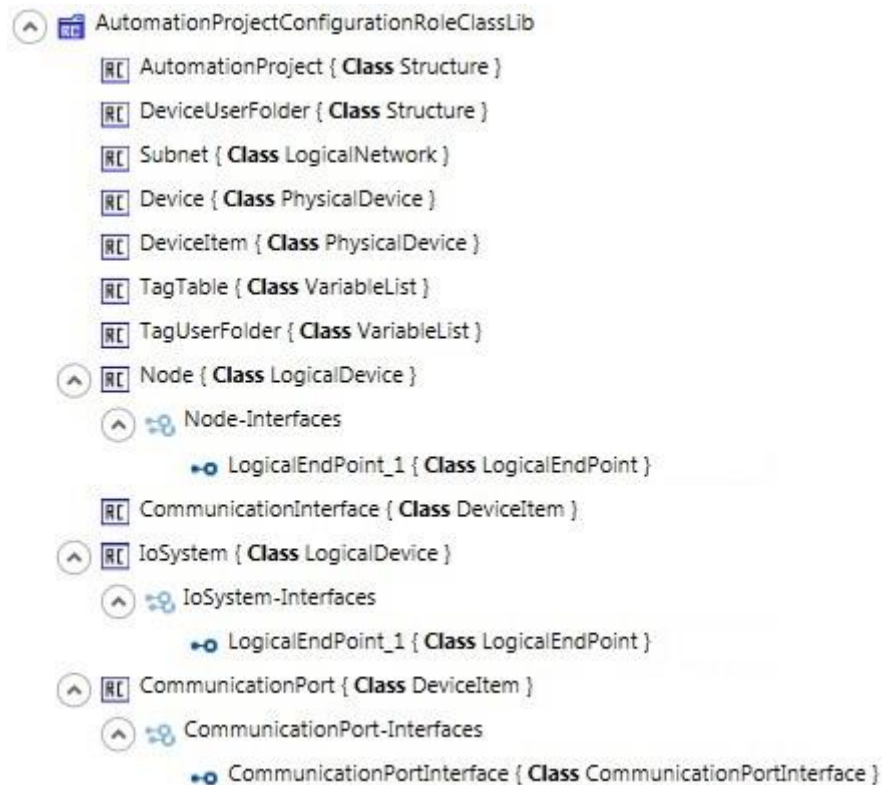


Figure 16 – AutomationProjectConfigurationRoleClassLib in AutomationML Editor view

```
<RoleClassLib Name="AutomationProjectConfigurationRoleClassLib">
  <Description>Automation Markup Language Automation Project Configuration Data Class Library</Description>
  <Version>0.9</Version>
  <RoleClass Name="AutomationProject"
    RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure">
    <Attribute Name="ProjectManufacturer" AttributeDataType="xs:string">
      <Description>manufacturer of the project</Description>
    </Attribute>
    <Attribute Name="ProjectSign" AttributeDataType="xs:string">
      <Description>unique identification of the project</Description>
    </Attribute>
    <Attribute Name="ProjectRevision" AttributeDataType="xs:string">
      <Description>revision number of the project</Description>
    </Attribute>
    <Attribute Name="ProjectInformation" AttributeDataType="xs:string">
      <Description>commenting information of the project</Description>
    </Attribute>
  </RoleClass>
  <RoleClass Name="DeviceUserFolder"
    RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure"></RoleClass>
```



```

<RoleClass Name="Subnet" RefBaseClassPath="CommunicationRoleClassLib/LogicalNetwork">
  <Attribute Name="Type" AttributeDataType="xs:string">
    <Description>type of the subnet</Description>
  </Attribute>
</RoleClass>
<RoleClass Name="Device" RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
    <Description>identifier of the device type</Description>
  </Attribute>
  <Attribute Name="Comment" AttributeDataType="xs:string">
    <Description>optional comment for the device</Description>
  </Attribute>
</RoleClass>
<RoleClass Name="DeviceItem" RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute Name="TypeName" AttributeDataType="xs:string">
    <Description>additional type information</Description>
  </Attribute>
  <Attribute Name="DeviceItemType" AttributeDataType="xs:string">
    <Description>classification of the DeviceItem</Description>
  </Attribute>
  <Attribute Name="PositionNumber" AttributeDataType="xs:int">
    <Description>slot number where this DeviceItem is plugged in</Description>
  </Attribute>
  <Attribute Name="BuiltIn" AttributeDataType="xs:boolean">
    <Description>define that this module is a build-in part of another module</Description>
    <DefaultValue>>false</DefaultValue>
  </Attribute>
  <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
    <Description>identifier of the DeviceItem type</Description>
    <DefaultValue>not applicable</DefaultValue>
  </Attribute>
  <Attribute Name="FirmwareVersion" AttributeDataType="xs:string">
    <Description>the firmware version of e.g. a CPU to identify the module correctly</Description>
  </Attribute>
  <Attribute Name="PlantDesignation IEC " AttributeDataType="xs:string">
    <Description>plant designation for this device item</Description>
  </Attribute>
  <Attribute Name="LocationIdentifier IEC " AttributeDataType="xs:string">
    <Description>location designation for this device item</Description>
  </Attribute>
  <Attribute Name="Comment" AttributeDataType="xs:string">
    <Description>optional comment for the device</Description>
  </Attribute>
  <Attribute Name="Address">
    <RefSemantic CorrespondingAttributePath="OrderedListType" />
    <Attribute Name="1">
      <Attribute Name="StartAddress" AttributeDataType="xs:string">
        <Description>start of the address</Description>
      </Attribute>
      <Attribute Name="Length" AttributeDataType="xs:string">
        <Description>total width of all of the channels on the device item</Description>
      </Attribute>
      <Attribute Name="IoType" AttributeDataType="xs:string">
        <Description>direction IN or OUT</Description>
      </Attribute>
    </Attribute>
  </Attribute>
</RoleClass>
<RoleClass Name="TagTable" RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice/VariableList" />
  <Attribute Name="AssignedToDefault" AttributeDataType="xs:boolean" />
</RoleClass>
<RoleClass Name="TagUserFolder" RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice/VariableList" />
<RoleClass Name="Node" RefBaseClassPath="CommunicationRoleClassLib/LogicalDevice">
  <Attribute Name="Type" AttributeDataType="xs:string">
    <Description>type of the network</Description>
  </Attribute>
  <Attribute Name="NetworkAddress" AttributeDataType="xs:string">
    <Description>network address of this device item</Description>
  </Attribute>
  <ExternalInterface Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="e3a58527-d133-49de-b2f4-23954fd5c13d" />

```

```

</RoleClass>
<RoleClass Name="CommunicationInterface"
  RefBaseClassPath="AutomationProjectConfigurationRoleClassLib/DeviceItem">
  <Attribute Name="Label" AttributeDataType="xs:string">
    <Description>name printed on the item</Description>
  </Attribute>
</RoleClass>
<RoleClass Name="IoSystem" RefBaseClassPath="CommunicationRoleClassLib/LogicalDevice">
  <Attribute Name="Number" AttributeDataType="xs:integer">
    <Description>unique number of the IOSystem</Description>
  </Attribute>
  <ExternalInterface Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="e6f2f4f0-e73c-4077-8856-785c50c3e008" />
</RoleClass>
<RoleClass Name="CommunicationPort" RefBaseClassPath="AutomationProjectConfigurationRoleClassLib/DeviceItem">
  <Attribute Name="Label" AttributeDataType="xs:string">
    <Description>name printed on the Port</Description>
  </Attribute>
  <ExternalInterface Name="CommunicationPortInterface"
    RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/CommunicationPortInterface"
    ID="3d5aa531-e435-4a8b-b4e7-49326359842f">
    <Attribute Name="Label" AttributeDataType="xs:string" />
  </ExternalInterface>
</RoleClass>
</RoleClassLib>

```

Figure 17 – AutomationProjectConfigurationRoleClassLib as XML representation

5.1.1 AutomationProject

An “AutomationProject” is derived from a “Structure” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 2 – Definition AutomationProject

Role class name	AutomationProject	
Description	The role class “AutomationProject” shall be used in order to represent the project from which the export arises.	
Parent Class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/AutomationProject	
Attributes	“ProjectName” (AttributeDataType="xs:string")	The attribute “ProjectName” shall define the name of the project. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	“ProjectManufacturer” (AttributeDataType="xs:string")	The attribute “ProjectManufacturer” shall define the manufacturer of the project.
	“ProjectSign” (AttributeDataType="xs:string")	The attribute “ProjectSign” shall define the unique identification of the project.
	“ProjectRevision” (AttributeDataType="xs:string")	The attribute “ProjectRevision” shall define the revision number of the project.
	“ProjectInformation” (AttributeDataType="xs:string")	The attribute “ProjectInformation” shall define commenting information of the project.

5.1.2 DeviceUserFolder

A “**DeviceUserFolder**” is derived from a “Structure” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 3 – Definition DeviceUserFolder

Role class name	DeviceUserFolder	
Description	The role class “DeviceUserFolder” shall be used in order to support the structure of a device within a project.	
Parent Class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/DeviceUserFolder	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” shall define the name of the DeviceUserFolder. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>

5.1.3 Subnet

A “**Subnet**” is derived from a “LogicalNetwork” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 4 – Definition Subnet

Role class name	Subnet	
Description	The role class “Subnet” shall be used in order to represent storing and managing of properties and functionality of networks.	
Parent Class	CommunicationRoleClassLib/LogicalNetwork	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Subnet	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” shall define the name of the subnet. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	“Type” (AttributeDataType=“xs:string”)	The attribute “Type” shall define the identifier of the type of the subnet

Note: Each SystemUnitClass or InternalElement having the role Subnet shall have at least one internal element with the role LogicalConnection coming from the role class library CommunicationRoleClassLib defined in AutomationML Whitepaper – Communication which contains an interface derived from the interface class logicalEndPoint coming from the interface class library CommunicationInterfaceClassLib defined in AutomationML Whitepaper – Communication. The interface shall be applied to link node elements and IoSystem elements to a subnet.

5.1.4 Device

A “**Device**” is derived from a “PhysicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 5 – Definition Device

Role class name	Device	
Description	The role class “Device” shall be used in order to represent a collection in which the individual HW objects of a slave or rack, including the slave or rack HW item, are brought together.	
Parent Class	CommunicationRoleClassLib/PhysicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Device	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” shall define the name of the device. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	“TypeIdentifier” (AttributeDataType=“xs:string”)	The attribute “TypeIdentifier” shall define the identifier of the device type
	“Comment” (AttributeDataType=“xs:string”)	The attribute “Comment” shall define an optional comment for the device

Note: The attribute “TypeIdentifier” shall have a prefix which describes the semantic of the following identifier separated by “:.” The following prefixes are allowed: “OrderNumber”, “GSD”, “System”, „CSP+“

Examples:

TypeIdentifier = “OrderNumber:3RK1 200-0CE00-0AA2”

TypeIdentifier = “GSD:SIEM8139.GSD/DAP”

TypeIdentifier = “System:Rack.Generic”

TypeIdentifier = “CSP+:AJ65VBTCE2-8T”

5.1.5 DeviceItem

A “**DeviceItem**” is derived from a “PhysicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 6 – Definition DeviceItem

Role class name	DeviceItem			
Description	The role class “DeviceItem” shall be used in order to represent an “abstract” object class for HW modules and submodules.			
Parent Class	CommunicationRoleClassLib/PhysicalDevice			
Path for Element reference	AutomationProjectConfigurationRoleClassLib/DeviceItem			
Attributes	“Name” (AttributeDataType=”xs:string”)		The attribute “Name” shall define the name of the device. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>	
	“TypeName” (AttributeDataType=”xs:string”)		The Attribute “TypeName” shall define additional type information.	
	“DeviceItemType” (AttributeDataType=”xs:string”)		The attribute “DeviceItemType” shall define the classification of the DeviceItem (e.g. CPU, HeadModule, Accessory).	
	“PositionNumber” (AttributeDataType=”xs:int”)		The attribute “PositionNumber” shall define the slot number where this DeviceItem is plugged in.	
	“BuiltIn” (AttributeDataType=”xs:boolean”)		The attribute “BuiltIn” shall define that this module is a build-in part of another module.	
	“TypeIdentifier” (AttributeDataType=”xs:string”)		The attribute “TypeIdentifier” shall define the identifier of the DeviceItem type.	
	“FirmwareVersion” (AttributeDataType=”xs:string”)		The attribute “FirmwareVersion” shall define the firmware version of e.g. a CPU to identify the module correctly.	
	“Comment” (AttributeDataType=”xs:string”)		The attribute “Comment” shall define an optional comment for the device	
	“Address” (OrderedListType)[0 ..n]	“1”	“StartAddress” (AttributeDataType=”xs:int”)	The attribute “StartAddress” shall define the start of the address.
			“Length” (AttributeDataType=”xs:int”)	The optional attribute “Length” shall define the total width of all of the channels on the device item.
			“IoType” (AttributeDataType=”xs:string”)	The attribute “IoType” shall specify the direction IN or OUT.
“PlantDesignation IEC” (AttributeDataType=” xs:string” and RefSemantic=” IEC81346”)		The attribute “PlantDesignation IEC” shall define the plant designation for this device item		
“LocationIdentifier IEC” (AttributeDataType=” xs:string” and RefSemantic=” IEC81346”)		The attribute “LocationIdentifier IEC” shall define the location designation for this device item		

Note: The Address attribute is a finite list of variable length. The number of list elements is application case dependent.

Note: The attributes PlantDesignation and LocationIdentifier are examples of reference designations following IEC 81346 and are not mandatory for the role class.

Note: The attribute "TypeIdentifier" shall have a prefix which describes the semantic of the following identifier separated by ":" The following prefixes are allowed: "OrderNumber", "GSD", "System", "CSP+".

Examples:

TypeIdentifier = "OrderNumber:3RK1 200-0CE00-0AA2"

TypeIdentifier = "GSD:SIEM8139.GSD/DAP"

TypeIdentifier = "System:Rack.Generic"

TypeIdentifier = "CSP+:AJ65VBTCE2-8T"

5.1.6 TagTable

A **"TagTable"** is derived from a "VariableList" according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 7 – Definition TagTable

Role class name	TagTable	
Description	The role class "TagTable" shall be used in order to support the structure of tags	
Parent Class	CommunicationRoleClassLib/PhysicalDevice/VariableList	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/TagTable	
Attributes	"Name" (AttributeDataType="xs:string")	The attribute "Name" shall define the name of the TagTable. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	"AssignToDefault" (AttributeDataType="xs:boolean")	The Attribute "AssignToDefault" shall define if the Tags inside will be imported to an existing default TagTable.

Note: While importing if the TagTable has an attribute with 'True' value, then all the Tags inside will be imported to an existing default TagTable. In this case the Name of the TagTable is ignored by the importing tool. By default, False value is assumed for 'AssignToDefault' attribute if it does not exist while importing.

5.1.7 TagUserFolder

A “**TagUserFolder**” is derived from a “VariableList” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 8 – Definition TagUserFolder

Role class name	TagUserFolder	
Description	The role class “TagUserFolder” shall be used in order to support the structure TagTables within a DeviceItem.	
Parent Class	CommunicationRoleClassLib/PhysicalDevice/VariableList	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/TagUserFolder	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” shall define the name of the TagUserFolder. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>

5.1.8 Node

A “**Node**” is derived from a “logicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 9 – Definition Node

Role class name	Node	
Description	The role class “Node” shall be used in order to specify all the interface related networking information of a network node.	
Parent Class	CommunicationRoleClassLib/logicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Node	
Attributes	“Type” (AttributeDataType=“xs:string”)	The attribute “Type” shall define the type of the network.
	“NetworkAddress” (AttributeDataType=“xs:string”)	The attribute “NetworkAddress” shall define the network address of this device item.
Interfaces	“LogicalEndPoint” (RefBaseClassPath=“CommunicationInterfaceClassLib/LogicalEndPoint”)	This interface shall be used to link the node to a subnet.

5.1.9 CommunicationInterface

A “**CommunicationInterface**” is derived from a “DeviceItem” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 10 – Definition CommunicationInterface

Role class name	CommunicationInterface	
Description	The role class “CommunicationInterface” shall be used in order to define the connection point of a device to a network.	
Parent Class	AutomationProjectConfigurationRoleClassLib/DeviceItem	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/CommunicationInterface	
Attributes	“Label” (AttributeDataType=“xs:string”)	The attribute “Label” shall define the name printed on the item.

5.1.10 IoSystem

An “**IoSystem**” is derived from LogicalDevice. It is defined as follows.

Table 11 – Definition IoSystem

Role class name	IoSystem	
Description	The role class “IoSystem” shall be used in order to model the master – slave relationship typically found in fieldbus systems.	
Parent Class	CommunicationRoleClassLib/LogicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/IoSystem	
Attributes	“Number” (AttributeDataType=“xs:int”)	The attribute “Number” shall define the unique number of the IoSystem.
Interfaces	“LogicalEndPoint” (RefBaseClassPath=“CommunicationInterfaceClassLib/LogicalEndPoint”)	This interface shall be used to link the IoSystem to a subnet.

5.1.11 CommunicationPort

A “**CommunicationPort**” is derived from a PhysicalEndpoint. It is defined as follows.

Table 12 – Definition CommunicationPort

Role class name	CommunicationPort	
Description	The role class “CommunicationPort” shall be used in order to model the device item applied to physically establish the connection to the network.	
Parent Class	AutomationProjectConfigurationRoleClassLib/PhysicalEndpoint	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/CommunicationPort	
Attributes	“Label” (AttributeDataType=“xs:string”)	The attribute “Label” shall define the name printed on the Port.
Interfaces	“CommunicationPortInterface” (RefBaseClassPath=“AutomationProjectConfigurationInterfaceClassLib/CommunicationPortInterface”)	This interface shall be used to link the Port to wires.

5.2 InterfaceClassLibrary

Second main basement of the modelling are the required interface classes. Facing the required model elements there are interface classes especially required for Automation Project Configuration data modelling derived from interface classes used from communication system modelling defined in AutomationML Whitepaper – Communication or derived from AutomationML basic interface classes defined in AutomationML Whitepaper – Architecture and general requirements

The following figures represent the interface class library.

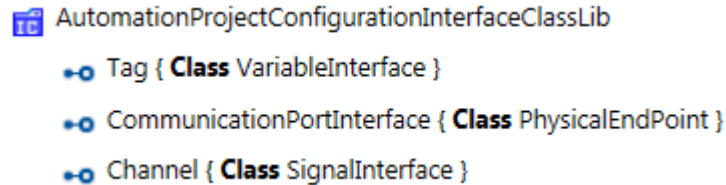


Figure 18 – AutomationProjectConfigurationInterfaceClassLib in AutomationML Editor view

```

<InterfaceClassLib Name="AutomationProjectConfigurationInterfaceClassLib">
  <Version>0.9</Version>
  <InterfaceClass Name="Tag" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/
    ExternalDataConnector/PLCopenXMLInterface/VariableInterface">
    <Attribute Name="DataType" AttributeDataType="xs:string">
      <Description>the type of the data (e.g. bool, byte, word)</Description>
    </Attribute>
    <Attribute Name="LogicalAddress" AttributeDataType="xs:string">
      <Description>address of the tag</Description>
    </Attribute>
    <Attribute Name="Comment" AttributeDataType="xs:string">
      <Description>optional comment for the device</Description>
    </Attribute>
  </InterfaceClass>
  <InterfaceClass Name="CommunicationPortInterface"
    RefBaseClassPath="CommunicationInterfaceClassLib/PhysicalEndPoint">
  </InterfaceClass>
  <InterfaceClass Name="Channel" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/
    Communication/SignalInterface">
    <Attribute Name="Type" AttributeDataType="xs:string">
      <Description>analog or digital type of the channel</Description>
    </Attribute>
    <Attribute Name="IoType" AttributeDataType="xs:string">
      <Description>direction IN or OUT</Description>
    </Attribute>
    <Attribute Name="Number" AttributeDataType="xs:int">
      <Description>number of the channel starting with 0</Description>
    </Attribute>
    <Attribute Name="Length" AttributeDataType="xs:integer">
      <Description>total width of the channel</Description>
    </Attribute>
  </InterfaceClass>
</InterfaceClassLib>
  
```

Figure 19 – AutomationProjectConfigurationInterfaceClassLib as XML representation

5.2.1 Tag

A “**Tag**” is derived from a “VariableInterface” according to AutomationML Whitepaper - Logic. It is defined as follows.

Table 13 – Definition Tag

Role class name	Tag	
Description	The Interface class “Tag” shall be used in order to represent the symbolic name of an I/O date. Tags shall only be used within a TagTable.	
Parent Class	VariableInterface	
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/Tag	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” shall define the name of the tag. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	“DataType” (AttributeDataType=“xs:string”)	The attribute “DataType” shall define the type of the data (e.g. BOOL, BYTE, WORD).
	“LogicalAddress” (AttributeDataType=“xs:string”)	The attribute “Logical Address” shall specify the address of the tag. <i>Note: The exporting ECAD tool defines the language mnemonic of the attribute. The importing PLC tool may change this mnemonic.</i> <i>The exporting PLC tool may follow the international mnemonic. The importing ECAD tool doesn’t change this mnemonic.</i> <i>Therefore in case of round trip engineering the use of the international or independent mnemonic in all participating tools is recommended.</i>
	“Comment” (AttributeDataType=“xs:string”)	The attribute “Comment” shall define an optional comment for the device

Tags without assigned channels and channels without assigned tags are possible (incomplete engineering)

5.2.1.1 CommunicationPortInterface

A “**CommunicationPortInterface**” is derived from a “PhysicalEndPoint” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 14 – Definition CommunicationPortInterface

Role class name	CommunicationPortInterface
Description	The interface class “CommunicationPortInterface” shall be used in order to define the physical connection to the network.
Parent Class	PhysicalEndPoint
Path for Element reference	

5.2.2 Channel

A “**Channel**” is derived from a “SignalInterface” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 15 – Definition Channel

Role class name	Channel	
Description	The role class “Channel” shall be used in order to define the process interface. A channel shall only be used within a DeviceItem.	
Parent Class	SignalInterface	
Path for Element reference		
Attributes	“Type” (AttributeDataType=“xs:string”)	The attribute “Type” shall define the analog or digital type of the channel (e.g. “Digital”, “Analog”).
	“IoType” (AttributeDataType=“xs:string”)	The attribute “IoType” shall specify the direction (e.g. “Input”, “Output”).
	“Number” (AttributeDataType=“xs:int”)	The attribute “Number” shall specify the number of the channel starting with 0.
	“Length” (AttributeDataType=“xs:int”)	The attribute “Length” shall define the total width of the channel.

A channel references with an Internal Link the associated Tags which are stored at a CPU DeviceItem.

5.2.3 Naming and Escaping

For all CAEX-Path Expressions the CAEX naming and escaping rules are defined as follows:

- In a name within a path contains the characters “[“ and “]” these characters have to be escaped by replacing them with “\[“ and “\]”
(e.g.: „R1/R1.[1]/R1.1.1“ => [R1]/[R1.\[1\]]/[R1.1.1]).
- If one of the characters “@”, “.”, “:” or “/” appears in the value of a path each part of the path must be enclosed by square brackets (“[“ and “]”). Example: RefPartnerSideA=“[4EA36EB0-8159-4829-8F4B-A829F3320E27]:[My.Tag]”
- Values of name attributes are not affected.
- Semantics of the operators “@” and “.” is not applied and must not be supported.

6 Practical examples

In this chapter practical examples for representation of Automation Project Configuration structures in AutomationML are shown to give a better understanding for the method displayed above.

6.1 Running example

The running example is established by parts of a lab size production system hosted at Otto-von-Guericke University Magdeburg. This production system contains a set of turntables, conveyers and multipurpose machines required to manufacture products. These transport systems and machines contain sensors and actors which are wired to control devices set up by intelligent field bus couplers from different vendors. The fieldbus couples are connected via Ethernet based communication to PC systems. This structure is depicted in Figure 20.

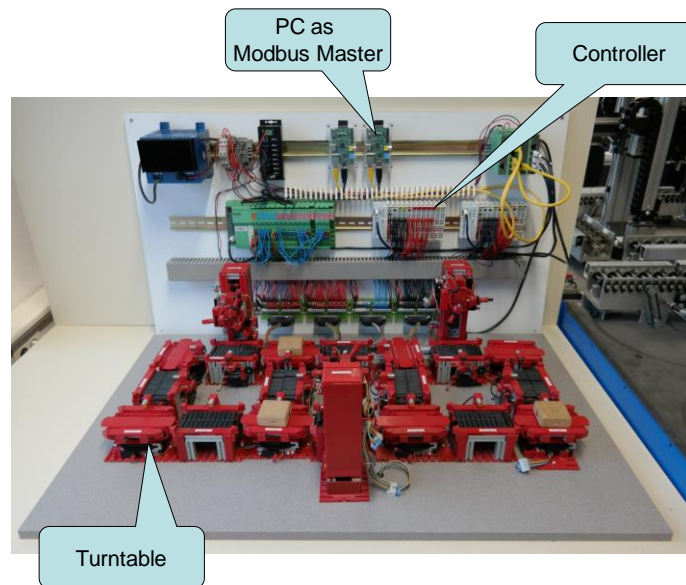


Figure 20 – Example system

Out of the example system only a small portion is considered in the example. This small portion contains a drive and an inductive sensor contained in a turntable. They are wired to an intelligent WagoFieldIO containing four input and output sub-devices as well as an intelligent fieldbus head. The fieldbus head is connected by an Ethernet cable and a Modbus TCP communication to the PC. Here the PC will act as Master in the Modbus communication. This structure is depicted in Figure 21.

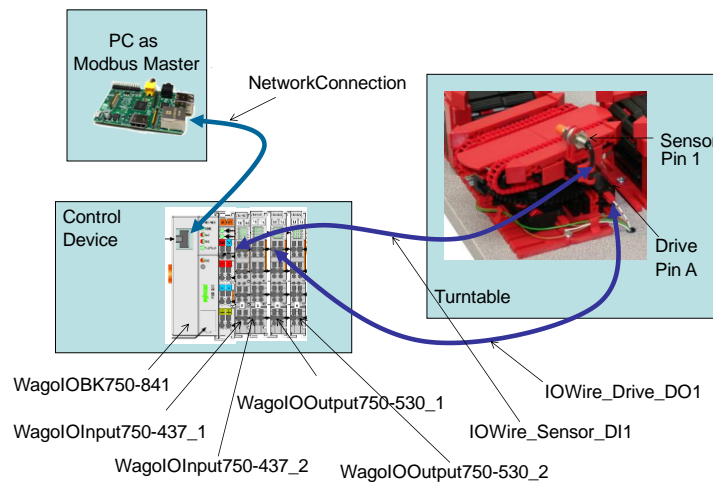


Figure 21 – Example system schematic representation

6.2 Additionally applied role classes

Following AutomationML Whitepaper – Communication for Automation Project Configuration modelling the following role classes will be applied in addition to the role classes defined in clause 5.1.

The following Figure shows the CommunicationRoleClassLib defined in AutomationML Whitepaper – Communication to be used.

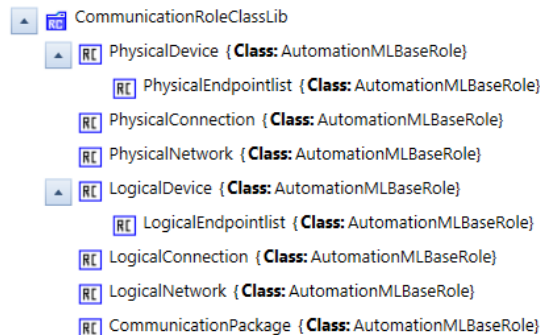


Figure 22 – CommunicationRoleClassLib

Out of this role class library the role classes LogicalConnection and PhysicalConnection need to be applied. They are used to indicate elements modelling the communication between network nodes or IOsystems within a subnet (LogicalConnection) and the physical wiring between Channels (PhysicalConnection).

6.3 Additionally applied interface classes

Following AutomationML Whitepaper – Communication for Automation Project Configuration modelling the following interface classes will be applied.

The following Figure shows the CommunicationInterfaceClassLib defined in AutomationML Whitepaper – Communication to be used.

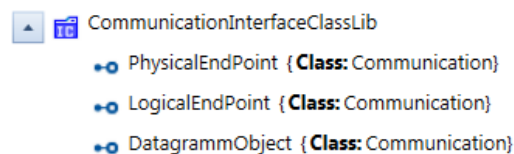


Figure 23 – CommunicationInterfaceClassLib

Out of this interface class library the interface classes LogicalEndPoint and PhysicalEndPoint need to be applied. They are used to model the linking points modelling the communication between network nodes or IOsystems within a subnet (LogicalEndPoint) and the physical wiring between Channels (PhysicalEndPoint).

6.4 SystemUnitClassLibrary

The SystemUnitClassLibrary to the described model is based on the definition of the necessary devices and connection elements relevant for the Automation Project Configuration. For the example these are:

- (1) the network card applied in the field bus coupler and the PC,
- (2) the digital input module,
- (3) the digital output module,
- (4) the programmable fieldbus controller including a network card (1),
- (5) the modular fieldbus I/O system containing a programmable fieldbus controller (4), two digital input modules (2) and two digital output modules (3) ,

- (6) the sensor,
- (7) the drive,
- (8) the wire,
- (9) the network,
- (10) the master slave communication network, and
- (11) the PC including a network card (1).

The elements of the SystemUnitClassLibrary are depicted in Figure 24.

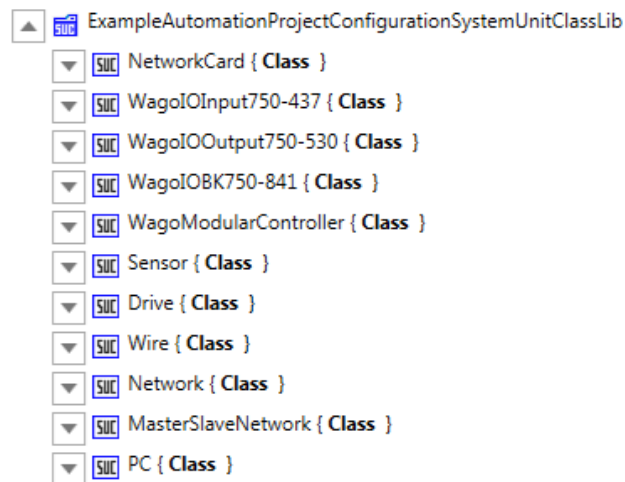


Figure 24 – SystemUnitClassLibrary of the example

In the following the different system unit classes are described in more detail.

6.4.1 Network card

The SystemUnitClass NetworkCard provides a model of generic network cards applicable within the PC as well as the Fieldbus coupler. It contains internal elements for the network node, the IOSystem, and the Port with relevant attributes and interfaces as defined in Chapter 5.1. It has the role CommunicationInterface and, therefore, a set of attributes defined in Section 5.1.9. The model is depicted in the following figure.

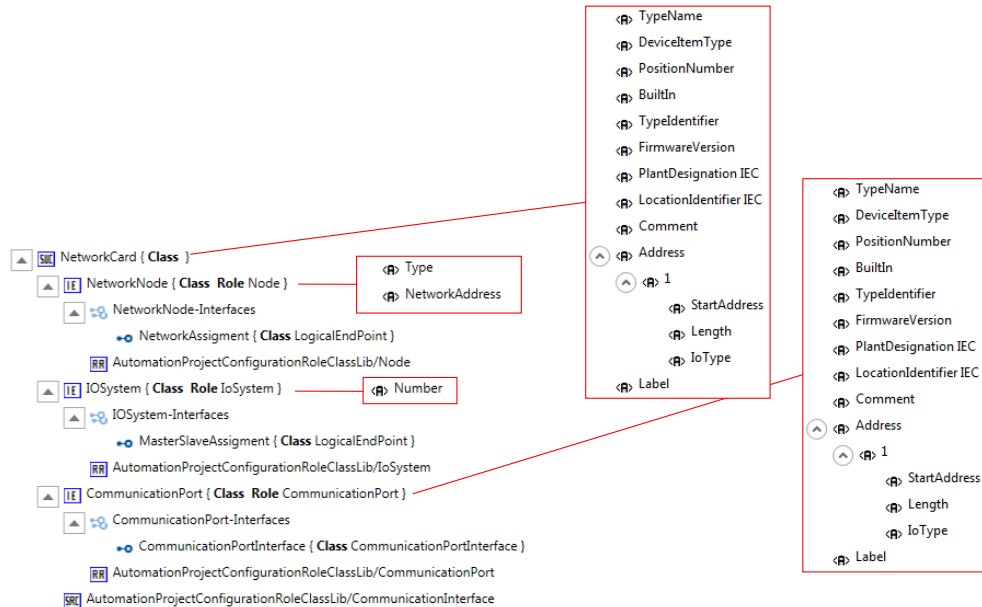


Figure 25 – Network card model

6.4.2 Digital input module

The SystemUnitClass WagoIOInput750-437 provides a model of an 8 digit binary input module applicable to model a modular field-IO. It contains a set of 8 channel interfaces for each input pin. In addition it has the role DeviceItem and therefore the attributes defined in Section 0. The model is depicted in the following figure.

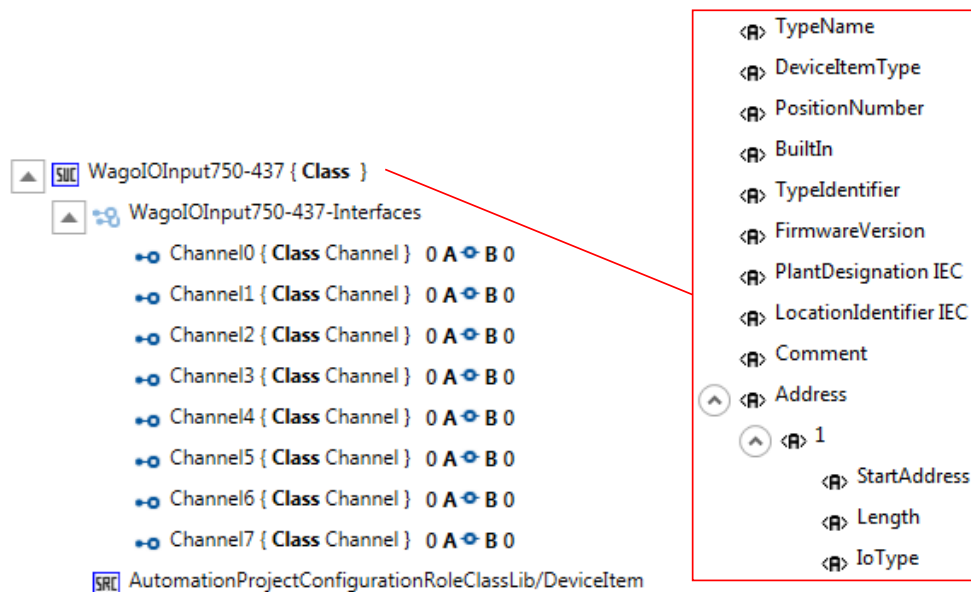


Figure 26 – Digital input module model

6.4.3 Digital output module

The SystemUnitClass WagoIOOutput750-530 provides a model of an 8 digit binary output module applicable to model a modular field-IO. It contains a set of 8 channel interfaces for each output pin. In addition it has the role DeviceItem and therefore the attributes defined in Section 0. The model is depicted in the following figure.

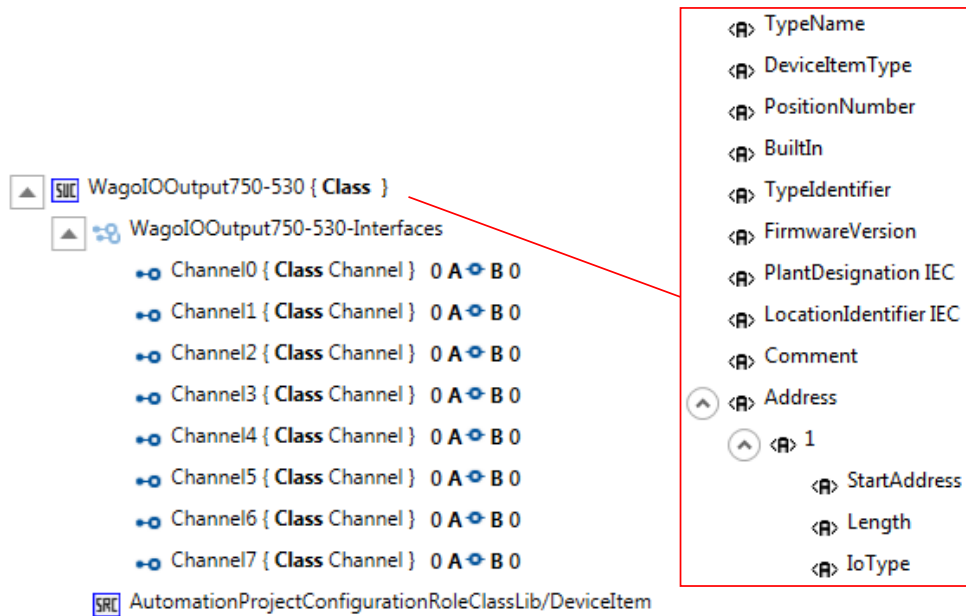


Figure 27 – Digital output module model

6.4.4 Programmable fieldbus controller

The SystemUnitClass WagoIOBK750-841 provides a model of a programmable fieldbus coupler applicable to model a modular field-IO. It contains InternalElements to model the TagUserFolders and the TagTables required for structuring the tags relevant within the Field-IO. They contain Tag interfaces. In addition the SystemUnitClass contains an InternalElement modelling the NetworkCard as defined in Chapter 5.1. Finally the SystemUnitClass has the role DeviceItem and therefore the attributes defined in Section 0. The model is depicted in the following figure.

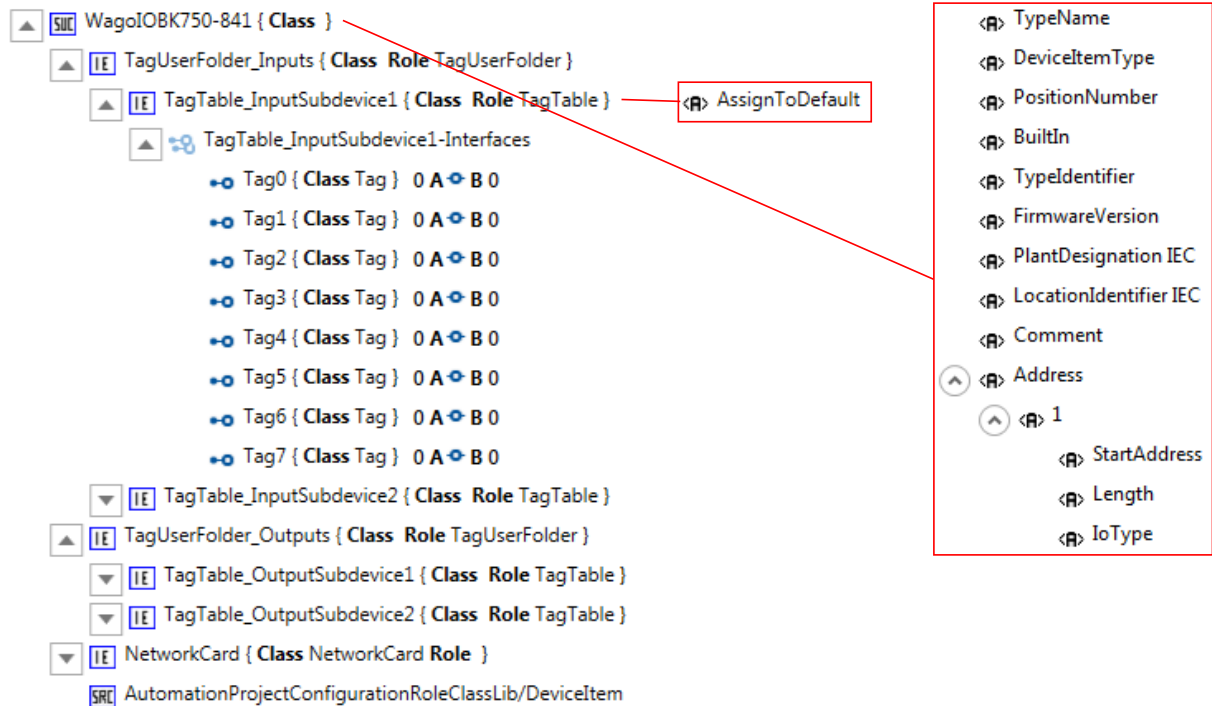


Figure 28 – Programmable fieldbus coupler model

6.4.5 Modular fieldbus I/O system

The SystemUnitClass WagoModularController models the modular fieldbus I/O system of the example with 2 input modules and 2 output modules. It contains InternalElements derived from the SystemUnitClasses WagoIOBK750-841, WagoIOInput750-437, and WagoIOOutput750-530 to model the modular structure of the modular fieldbus I/O system. Within the SystemUnitClass related Channel and Tag interfaces are connected by using internal links. Thereby, the assignment of a variable to a physical input is modelled. Finally, the SystemUnitClass has the role Device and, therefore, the attributes defined in Section 5.1.4. The model is depicted in the following figure.

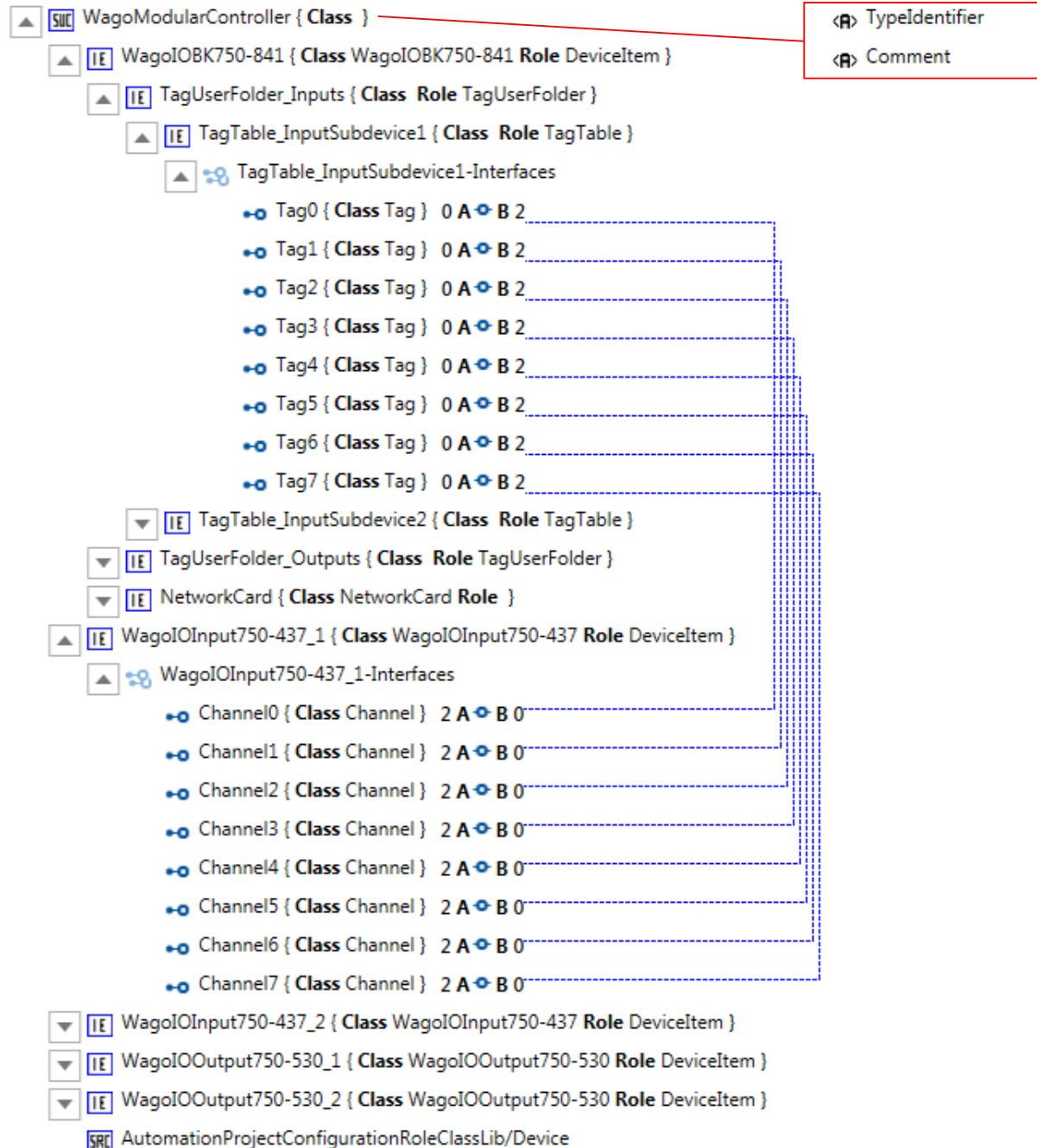


Figure 29 – Modular controller model

6.4.6 Sensor

The SystemUnitClass Sensor models the applied sensor within the example system. It contains one Channel interface to model the physical output to the sensor and one Tag interface to model the logical value of the sensor state. The SystemUnitClass has the role Device and, therefore, the attributes defined in Section 5.1.4. In addition the SystemUnitClass has the role Sensor from the AutomationMLCSRoleClassLib from AutomationML Part 2 to identify it is a sensor. The model is depicted in the following figure.

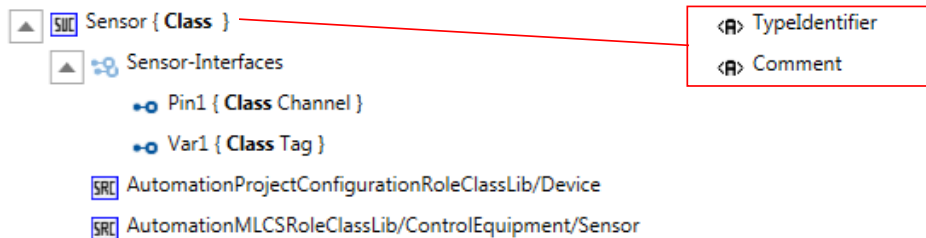


Figure 30 – Sensor model

6.4.7 Drive

The SystemUnitClass Drive models the applied actor within the example system. It contains one Channel interface to model the physical input to the actor and one Tag interface to model the logical value of the actor state. The SystemUnitClass has the role Device and, therefore, the attributes defined in Section 5.1.4. In addition the SystemUnitClass has the role Actuator from the AutomationMLCSRoleClassLib from AutomationML Part 2 to identify it is an actuator device. The model is depicted in the following figure.

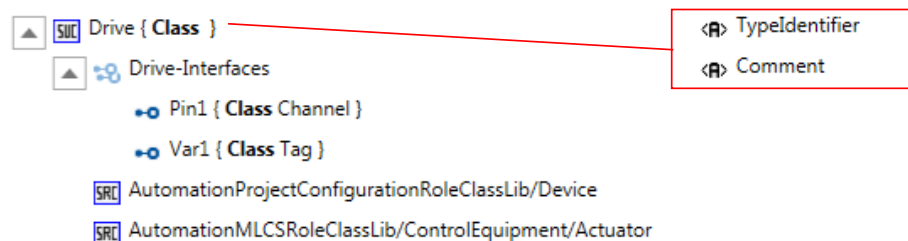


Figure 31 – Drive model

6.4.8 Wire

The SystemUnitClass Wire models the necessary wiring connections between the different channel interfaces following AutomationML Whitepaper – Communication. Therefore it contains two PhysicalEndPoint interfaces and has the role class PhysicalConnection. The model is depicted in the following figure.

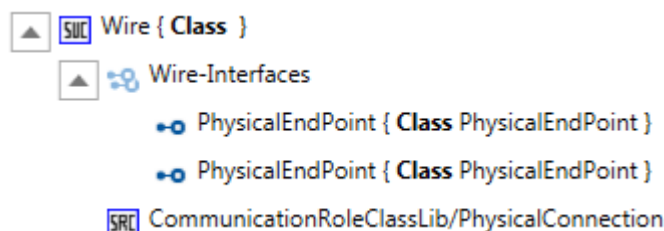


Figure 32 – Wire model

6.4.9 Network

The SystemUnitClass Network models the ModbusTCP communication network as an Ethernet network. Following AutomationML Whitepaper – Communication it contains one LogicalEndPoint interfaces applied to assign the different network nodes to the network by linking the related interfaces. The SystemUnitClass has the role class SubNet and, thereby, the related interfaces defined in Section 5.1.3. The model is depicted in the following figure.

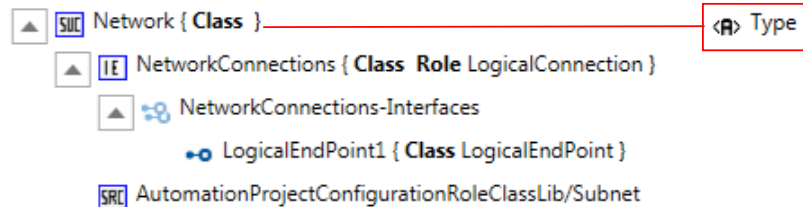


Figure 33 – Network model

6.4.10 Master slave communication network

The SystemUnitClass MasterSlaveNetwork models the Master-Slave relation of the ModbusTCP communication network. Following AutomationML Whitepaper – Communication it contains one LogicalEndPoint interfaces applied to assign the different IOSystem elements to the network by linking the related interfaces. The SystemUnitClass has the role class SubNet and, thereby, the related interfaces defined in Section 5.1.3. The model is depicted in the following figure.

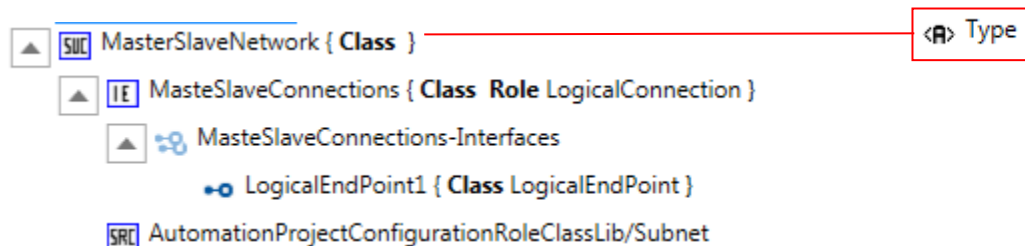


Figure 34 – Master slave communication network model

6.4.11 PC

The SystemUnitClass PC models the PC acting as Modbus TCP master. It contains an Internal Element modelling the network card following the definition above. In addition it has the role Device and, therefore, the attributes defined in Section 5.1.4. The model is depicted in the following figure.

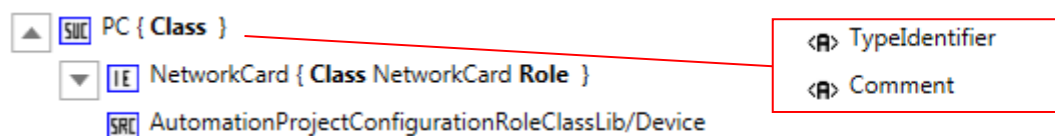


Figure 35 – PC model

6.5 InstanceHierarchy

Applying the SystemUnitClasses described in the last Section the instance hierarchy can be modelled.

Therefore the relevant modelling elements need to be embedded in the expected hierarchy of the manufacturing system structure applied. In the running example this system structure is based on the ISA 95 hierarchy of elements of a production system. Thus it models the hierarchy of an enterprise with sites which contain manufacturing areas in which different production lines exist consisting of work cells and units. All these layers are indicated by roles defined in the role class library AutomationMLExtendedRoleClassLib defined in AutomationML Part 2. In addition there are modelling elements for the enclosure and the wiring following the AutomationML whitepaper "Communication". The structure is depicted in Figure 36.

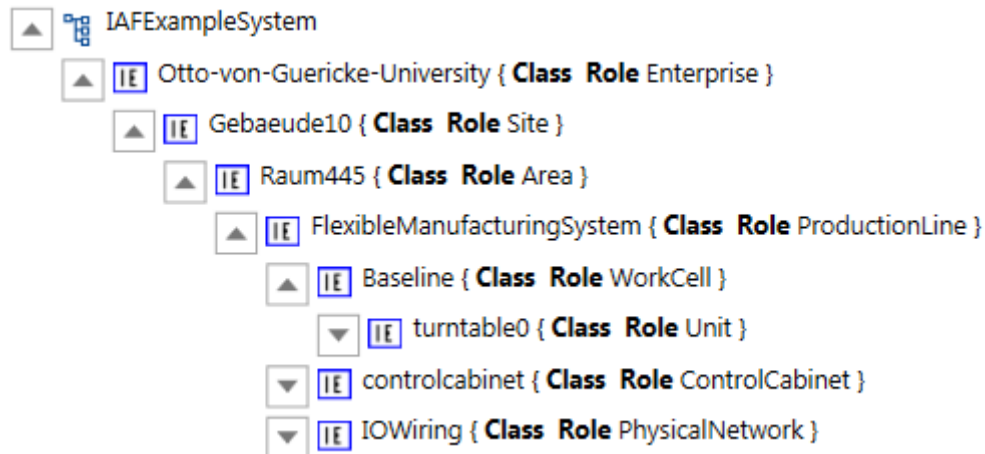


Figure 36 – Upper layer hierarchy elements

Within the defined hierarchy the control devices are embedded. In case of the running example there are a sensor and a drive InternalElement instantiated from the relevant system unit classes as depicted in Figure 37.

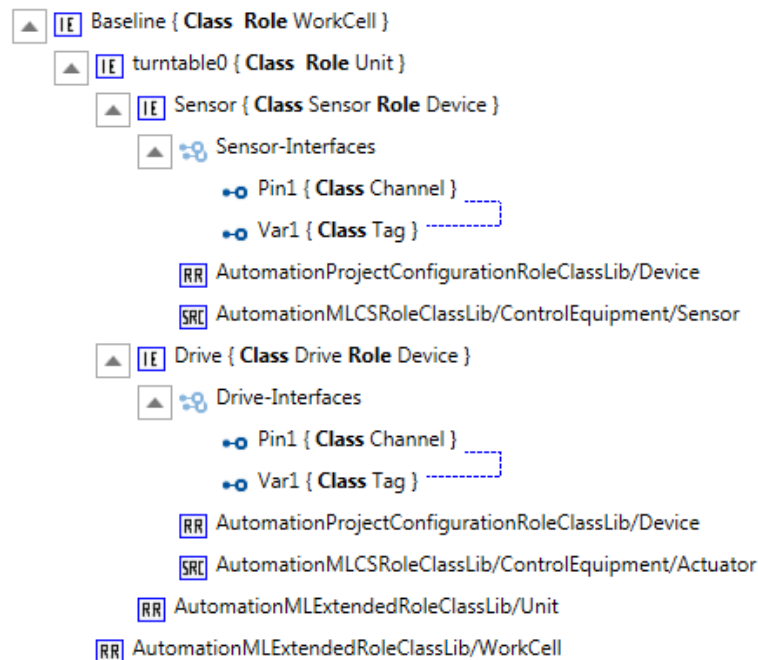


Figure 37 – Modelling elements of turntable

The modelling elements of the control devices within the control cabinet are integrated in the corresponding InternalElement. Therefore, on the one hand, the system unit classes WagoModularController, PC, Network, and MasterSlaveNetwork are instantiated as Internal Elements. On the other hand an InternalElement with the role PhysicalNetwork is modelled containing instances of the system unit class Wire to model the Ethernet wiring following AutomationML whitepaper "Communication". This structure is depicted in Figure 38.

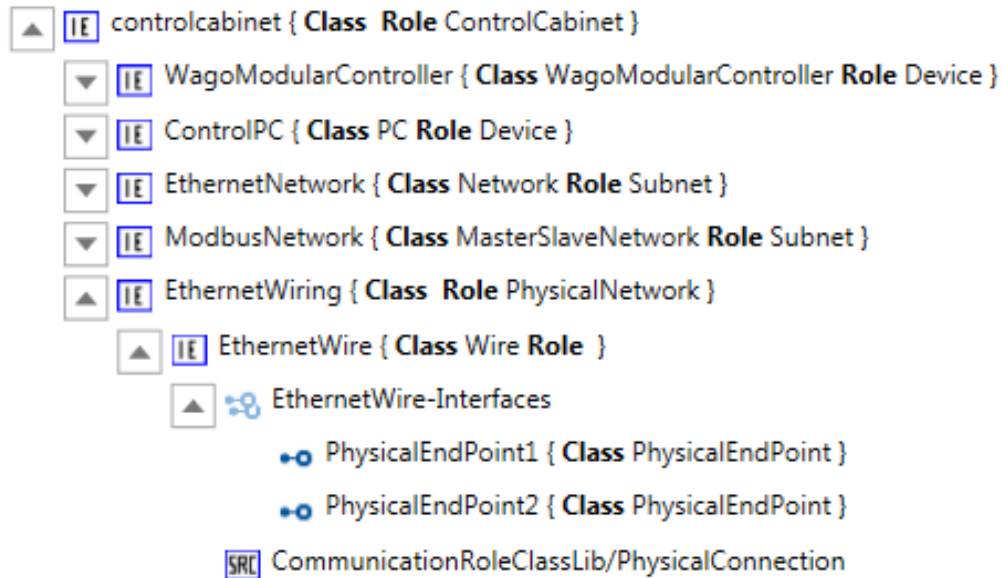


Figure 38 – Modelling elements of control cabinet

The wire objects are connected to the CommunicationPort objects of the PC and WagoIOBK750-841 objects. Therefore, the related interfaces of the CommunicationPortInterface type and PhysicalEndPoint type are linked by an internal link. Figure 39 depicts this link for the case of the WagoIOBK750-841 linked to the EthernetWire.

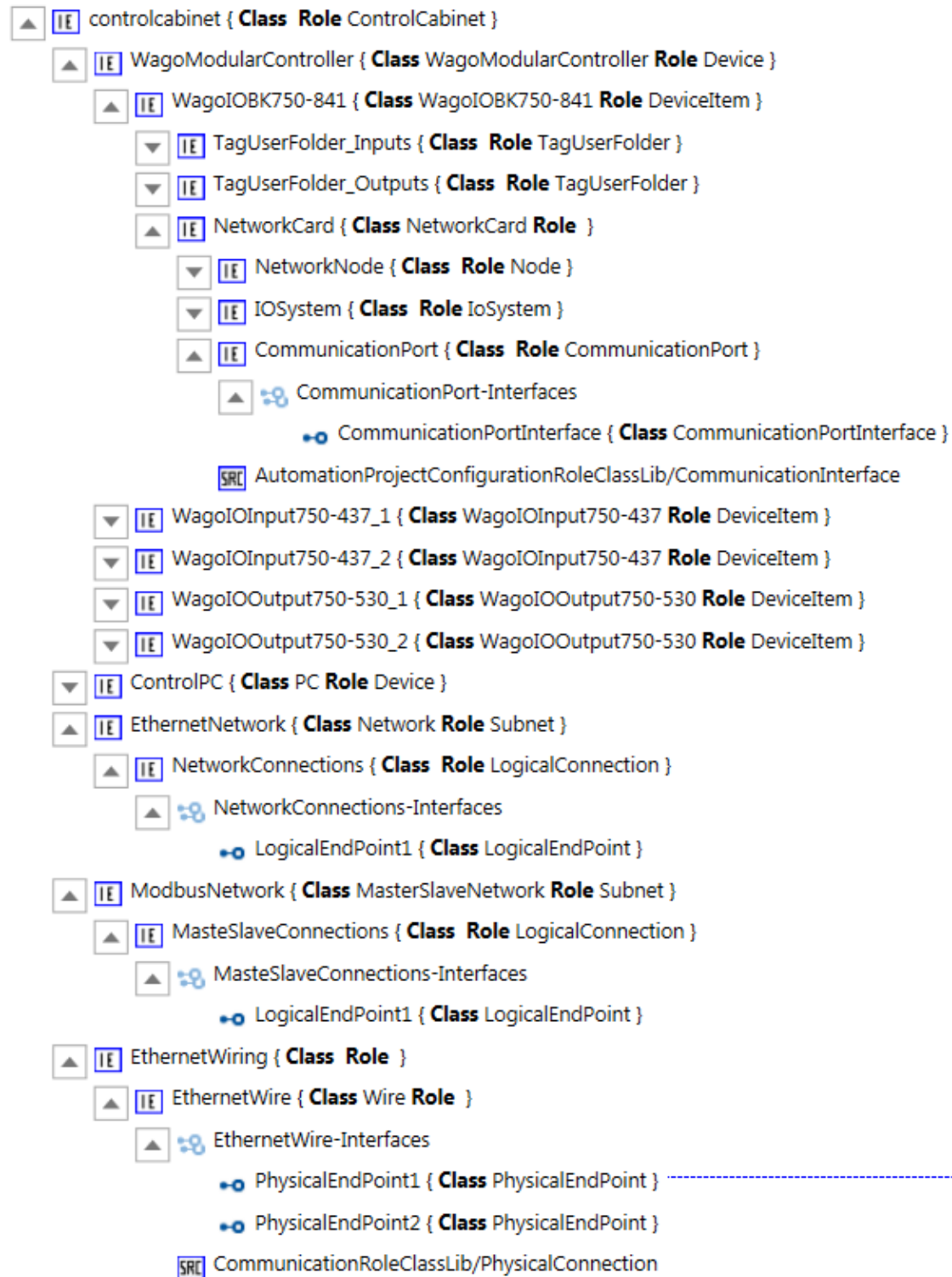


Figure 39 – Modelling of communication wiring based on wire elements and internal links

In addition to the Ethernet Wiring also the physical wiring of the control devices need to be modelled. Therefore, the system unit class Wire is instantiated two times with the InternalElement IOWiring as depicted in Figure 40. These wire objects are connected by linking the interfaces of the interface class PhysicalEndPoint on the one hand to the channel interfaces of the sensor and actor InternalElements as given in Figure 41 and on the other hand to the channel interfaces within the WagoIOBK750-841 InternalElement.

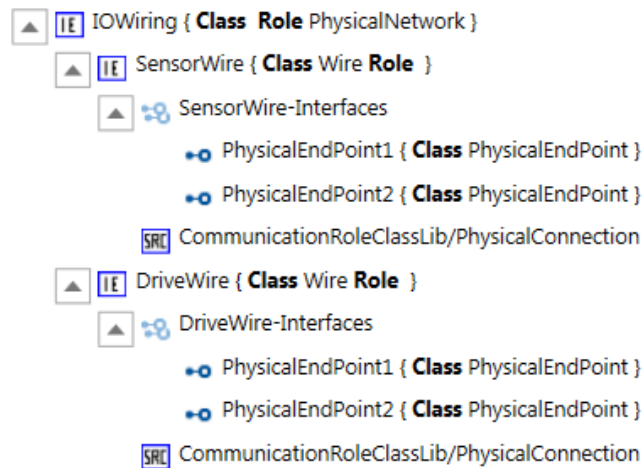


Figure 40 – Modelling elements of wiring

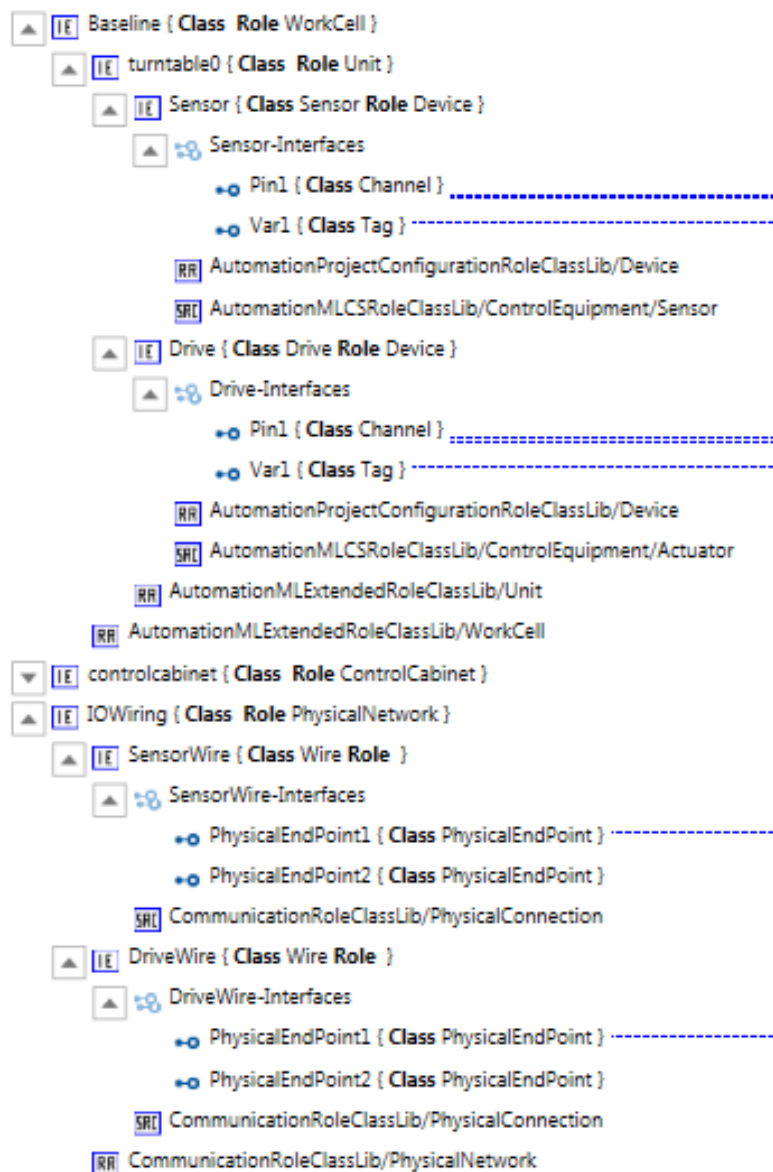


Figure 41 – Modelling of physical wiring based on wire objects and internal links

Finally the different devices need to be assigned to the different subnets and IOSystems. This is done by modelling InternalLinks between the Interfaces of the type logicalEndPoint within the InternalElements NetworkNode or IOSystem on the one side and Network or MasterSlaveNetwork on the other side. This structure is depicted in Figure 42.

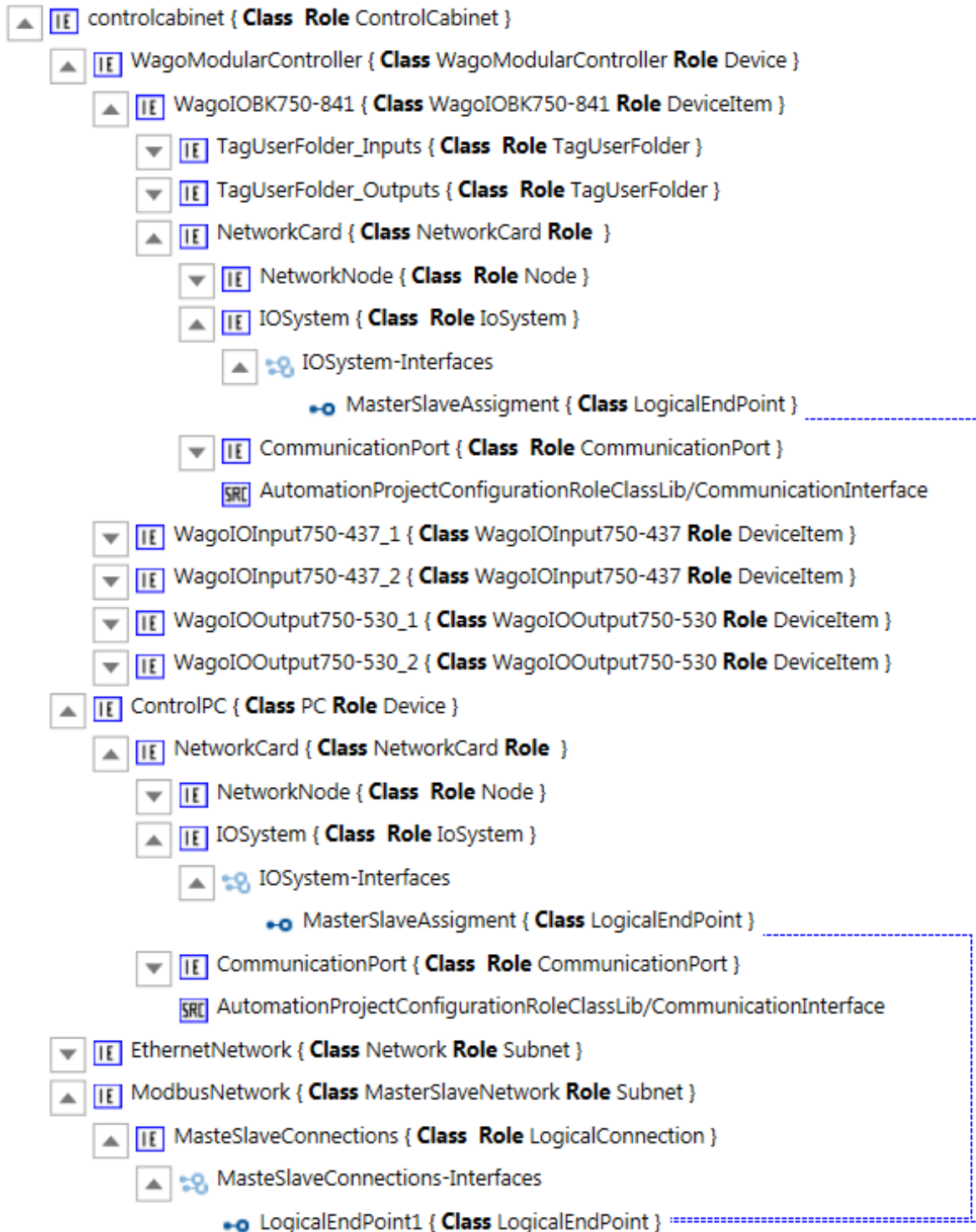


Figure 42 – Modelling of device assignment to networks

All instantiated elements of the model may get attribute values. For example the WagoModularController instance will get values for the attributes TypeIdentifier and Comment. Its sub-element WagoIOBK750-841 will get values for the attributes TypeName, DeviceType, PositionNumber, BuiltIn, and others. And, the Channel0 interface within the WagoIOInput750-437_1 InternalElement will have values for the attributes DataType, LogicalAddress, and Comment. This example (and further given attribute values) are depicted in Figure 43. It shall be recognised, that, following AutomationML part 1 – Architecture, not all attributes of the SystemUnitClass referenced by an InternalElement by RefBaseSystemUnitPath or RoleClasses referenced by SupportedRoleClass and/or RoleRequirement need to be provided with a value. Not used attributes can be deleted in the InstanceHierarchy.

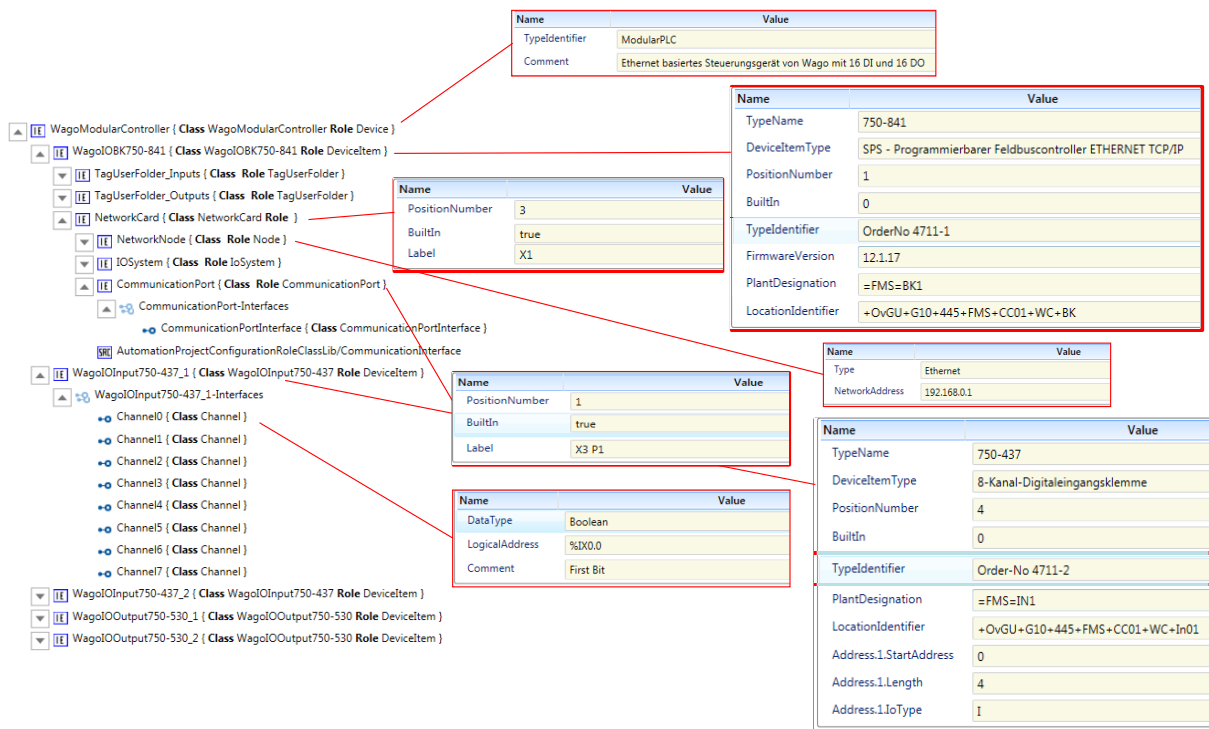


Figure 43 – WagoModularController instance with relevant parameters described by attributes