



<AutomationML/>

**The Glue for Seamless
Automation Engineering**

**Whitepaper AutomationML
Part 6: AutomationML Component
Document Identifier: WP Compo, V 1.1.0
State October 2020**

©AutomationML consortium

Version 1.0, October 2020

Contact: www.automationml.org

Content

Content	3
List of Figures	8
List of Tables	13
Contributor	15
Glossary	16
1 Introduction	18
1.1 Overview	18
1.2 Basics	19
1.3 Scope	20
1.4 References	20
2 Automation Component	22
2.1 Stakeholders and Lifecycle	23
2.2 Use Cases	26
2.2.1 Materials Management and Warehousing	26
2.2.2 Component Description as Base for xCAD	26
2.2.2.1 Electrical Engineering and software design	27
2.2.2.2 Mechanical Engineering	28
2.2.2.3 Fluidic Engineering	29
2.2.3 Simulation and Virtual Commissioning	29
2.2.4 Maintenance and Documentation	30
2.2.5 Device Description Files for Field Devices	32
2.3 Considered Information and sub-model of Automation Components	33
2.4 Views on Automation Components and Systems	34
2.5 AutomationML Base Technology CAEX 2.15 and CAEX 3.0	36
3 Representation in AutomationML	37
3.1 Structuring of AutomationML Components and Composite Components	37
3.2 Composition of AutomationML Automation Components	38
3.2.1 General Provisions	38
3.2.2 Example simple AutomationML Component	38
3.2.3 Example nested AutomationML Component with AutomationML Objects	39
3.2.4 Example AutomationML Composite Component	39
3.2.5 Example nested AutomationML Composite Component with hierarchies	40
3.3 Mapping of General Data	41
3.4 General Model Integration	42
3.5 AutomationML Component Logic Representation	43
3.5.1 General	43
3.5.2 Logic Models	44
3.5.3 Usage of Logic Models	47
3.5.4 Behaviour	49
3.5.4.1 PLCOpen XML based Behaviour Model	50
3.5.4.2 AMLLogic based Behaviour Model	51
3.5.4.3 FMI based Behaviour Model	52
3.5.5 Simulation	53
3.5.5.1 PLCOpen XML based Simulation Model	54

3.5.5.2	AMLogic based Simulation Model	54
3.5.5.3	FMI based Simulation Model	54
3.5.6	Sequencing	54
3.5.6.1	PLCopen XML based SequencingElements	56
3.5.6.2	AMLogic based SequencingElements	56
3.5.6.3	FMI based SequencingElements	56
3.5.7	Function	56
3.5.7.1	PLCopen XML based Function	57
3.5.7.2	AMLogic based Function	57
3.5.7.3	FMI based Function	57
3.5.8	SkillsLogic	57
3.5.8.1	PLCopen XML based SkillLogicModel	58
3.5.8.2	AMLogic based SkillLogicModel	58
3.5.8.3	FMI based SkillLogicModel	59
3.6	Geometry and Kinematic Model	59
3.6.1	Geometry	59
3.6.1.1	Collada Geometry Model	59
3.6.1.2	JT Model	61
3.6.1.3	2D Model	62
3.6.2	Kinematics	63
3.6.2.1	Collada Kinematic Model	64
3.7	Graphic Representation	69
3.8	Documentation	70
3.9	Certificates	71
3.10	Additional Device Description	71
3.11	Maintenance Description	72
3.12	Skill Description for AutomationML Components	73
3.13	Connector for AutomationML Components	74
3.13.1	General	74
3.13.2	Mechanic Connector	76
3.13.3	Logic Connector	77
3.13.4	Electric Connector	78
3.13.4.1	General	78
3.13.4.2	General modelling provisions for electric connectors	79
3.13.5	Fluidic Connector	81
3.13.6	Liquidic Connector	81
3.13.7	Pneumatic Connector	82
3.13.8	Hydraulic Connector	83
3.13.9	Sensor Connector	83
3.13.10	Skill Connector	84
3.13.11	Multi Connectors	84
3.14	Connecting AutomationML Components and Composite Components	85
3.14.1	Relations between the contents within an AutomationML Component	85
3.14.1.1	Relations between Connectors and Models within an AutomationML Component	86
3.14.1.2	Relations between Models within an AutomationML Component	86
3.14.2	Composite Components: Relations between different AutomationML Components	86
4	Standard Libraries	88
4.1	Role Class Libraries	88

4.1.1	Overview new Role Class Libraries	88
4.1.2	AutomationMLComponentBaseRCL	90
4.1.2.1	RoleClass AdditionalDeviceDescription	90
4.1.2.2	RoleClass Connector	91
4.1.2.3	RoleClass Documentation	91
4.1.2.4	RoleClass GeometryModel	92
4.1.2.5	RoleClass GraphicRepresentation	92
4.1.2.6	RoleClass Icon	93
4.1.2.7	RoleClass LogicModel	93
4.1.2.8	RoleClass PLCopenXMLLogic	94
4.1.2.9	RoleClass AMLLogic	95
4.1.2.10	RoleClass FMLogic	95
4.1.2.11	RoleClass KinematicModel	96
4.1.2.12	RoleClass MaintenanceDescription	96
4.1.2.13	RoleClass Model	97
4.1.2.14	RoleClass Symbol	97
4.1.3	AutomationMLComponentStandardRCL	98
4.1.3.1	RoleClass AutomationComponent	99
4.1.3.2	RoleClass AutomationComponentSemanticSystem	104
4.1.3.3	RoleClass BehaviourModel	106
4.1.3.4	RoleClass Function	106
4.1.3.5	RoleClass SimulationModel	106
4.1.3.6	RoleClass SkillLogicModel	107
4.1.3.7	RoleClass SequencingModel	107
4.1.3.8	RoleClass Sequence	107
4.1.3.9	RoleClass SequenceElement	108
4.1.3.10	RoleClass COLLADAKinematicModel	108
4.1.3.11	RoleClass COLLADAKinematicJoint	109
4.1.3.12	RoleClass COLLADAKinematicAttachment	109
4.1.3.13	RoleClass COLLADAGeometryModel	110
4.1.3.14	RoleClass COLLADAGeometryAttachment	110
4.1.3.15	RoleClass JTGeometryModel	111
4.1.3.16	RoleClass 2DGeometryModel	111
4.1.3.17	RoleClass ComponentPicture	112
4.1.3.18	RoleClass ElectricSymbol	112
4.1.3.19	RoleClass HydraulicSymbol	112
4.1.3.20	RoleClass PneumaticSymbol	113
4.1.3.21	RoleClass ManufactureIcon	113
4.1.3.22	RoleClass ComponentIcon	113
4.1.3.23	RoleClass SkillModel	114
4.1.3.24	RoleClass Certificate	114
4.1.3.25	RoleClass MechanicConnector	114
4.1.3.26	RoleClass LogicConnector	115
4.1.3.27	RoleClass ElectricConnector	115
4.1.3.28	RoleClass FluidicConnector	115
4.1.3.29	RoleClass LiquidicConnector	116
4.1.3.30	RoleClass PneumaticConnector	116
4.1.3.31	RoleClass HydraulicConnector	117
4.1.3.32	RoleClass SensorConnector	117
4.1.3.33	RoleClass SkillConnector	118
4.1.3.34	RoleClass MaintenanceDescriptionGroup	118
4.1.3.35	RoleClass MaintenanceDescriptionItem	119
4.1.4	AutomationMLFMLogicRoleClassLib	121
4.1.4.1	RoleClass FMLogicObject	121
4.2	Interface Class Libraries	122
4.2.1	Overview	122
4.2.2	AutomationMLComponentBaseICL	122

4.2.2.1	InterfaceClass GraphicRepresentationReference	123
4.2.2.2	InterfaceClass 2DReference	124
4.2.2.3	InterfaceClass JTRreference	124
4.2.2.4	InterfaceClass SkillInterface	124
4.2.2.5	InterfaceClass DeviceDescriptionReference	125
4.2.2.6	InterfaceClass MaintenanceDescriptionLink	125
4.2.2.7	InterfaceClass MechanicInterface	125
4.2.2.8	InterfaceClass ElectricInterface	126
4.2.2.9	InterfaceClass LiquidicInterface	126
4.2.2.10	InterfaceClass PneumaticInterface	126
4.2.2.11	InterfaceClass PneumaticConnector	127
4.2.2.12	InterfaceClass CondensateDrainConnector	127
4.2.2.13	InterfaceClass HydraulicInterface	128
4.2.2.14	InterfaceClass SensorInterface	128
4.2.2.15	InterfaceClass JointInterface	128
4.2.3	AutomationMLFMIInterfaceClassLib	129
4.2.3.1	InterfaceClass FMIRreference	129
4.2.3.2	InterfaceClass FMIVariableInterface	130
5	Exchange of AutomationML Components as AMLX Container	131
5.1	Use Cases for AMLX Container	131
5.2	AMLX Container types for AutomationML Components	131
5.2.1	AMLX Container for single AutomationML Component	131
5.2.2	AMLX Container for AutomationML Component catalogues	132
5.2.3	AMLX Container for single AutomationML Composite Component	133
6	Manufacturer Specific Extensions of Component Description	135
7	Semantics in Automation Components	136
7.1	Concept of Semantic System Integration	136
7.1.1	Definition of used Semantic Systems	136
7.1.2	Referencing Attributes Semantic	137
7.2	Example for Usage	139
8	Automation Components in CAEX 3.0	141
8.1	General	141
8.2	CAEX Schema Transformation	141
8.3	AutomationML Version Transformation	141
8.3.1	Exchange of used libraries	141
8.3.2	Adaptation of the component libraries	141
8.4	Modelling of electrical interfaces with CAEX 3.0	142
8.4.1	Whats new	142
8.4.2	Example: M12 interface class library	142
8.4.3	Example: Mini 7/8 interface class library	147
8.4.4	Example: RJ45 interface class library	147
8.4.5	Application role class library	148
8.4.6	Application Example: Automation component with multiple electric connectors	148
8.4.7	Application Example: M12 to M12 cable	149
9	Practical examples for Automation Components	151
9.1	Example of a typical Automation Component: Pneumatic Cylinder	151
9.1.1	Container Package	151
9.1.2	Model Overview as Object Tree	152
9.1.3	Connectors	153
9.1.4	Behaviour	153

9.1.5	Kinematics and Geometry	153
9.1.6	Attributes	154
9.1.7	Graphical Representation	154
9.1.8	Documentation	154
9.1.9	Summary	155
9.2	Skill Example based on VDMA SOArc and VDI 2860 standards	155
9.2.1	Generic model of a skill based on VDMA SOArc and VDI standards	155
9.2.2	Modelling the provided skill of a component/system based on VDMA SOArc and VDI 2860 standards	156
9.3	Connection of components	157
9.4	Motor and Frequency Converter	158
9.4.1	Example Motor	158
9.4.2	Example Frequency Converter	161
9.4.3	Motor / Drive / PLC	163
9.5	User Defined Simulation Models	165
9.5.1	Referencing Functional Mockup Units According to the FMI Standard	165
9.5.2	SIMIT Simulation Models	165
9.6	Additional Device Description	170
9.7	Geometry and Kinematic Example	171
9.7.1	Motor	171
9.7.2	Adapter	172
9.7.3	Electromechanical Drive	174
9.7.4	Interconnection of the components to a drive train	175
9.8	Semantic of AutomationML Component Attributes	176
9.9	Modelling of a library of electrical M12 connector types	176
9.9.1	General	176
9.9.2	M12 role class library	176
9.9.3	Example application role class library	177
9.9.4	Application Example: Automation component with multiple electric connectors	177
9.9.5	Application Example: M12 to M12 cable	178
9.9.6	Discussion	179
10	Variants of Automation Components in AutomationML	180

List of Figures

Figure 1: Aspects of an automation component or production system.....	22
Figure 2: Overview of tool use for AutomationML Components	23
Figure 3: Stakeholder of automation component	24
Figure 4: Data flow between stakeholders over the lifecycle.....	26
Figure 5: Use Case component description a base for xCAD	27
Figure 6: Schematic example of an FMI-based co-simulation model for VC including HIL.....	30
Figure 7: Use Case Maintenance and documentation	31
Figure 8: Principle of the AML-DD package with an example for an IO-Link device	33
Figure 9: Information and sub-model integration into to the AutomationML Component	33
Figure 10: Overview structure of an automation component in AutomationML	38
Figure 11: Example simple AutomationML Component	39
Figure 12: Example nested AutomationML Component with AutomationML Objects	39
Figure 13: Example AutomationML Composite Component	40
Figure 14: Example nested AutomationML Composite Component with hierarchies	41
Figure 15: Representation of attribute groups as AutomationML attributes as UML diagram	42
Figure 16: AutomationComponent role class with attribute group definition	42
Figure 17: Example an AutomationML Component with one “Model”	43
Figure 18: AutomationML Representation of “Model” role class	43
Figure 19: AutomationML Component logic models.....	44
Figure 20: Role Classes for automation component Logic Model	45
Figure 21: Interface Classes for automation component Logic Model	45
Figure 22: Inheritance structure for an AutomationML logic model integration	46
Figure 23: Role Classes for automation component logic use cases	47
Figure 24: Inheritance structure for an AutomationML logic use cases.....	48
Figure 25: Example integration of an FMI Logic Model	48
Figure 26: Inheritance structure for an AutomationML “BehaviourModel”	49
Figure 27: AutomationML Representation of “BehaviourModel” role class	50
Figure 28: Example usage of the AutomationML role class “PLCLogic” and “BehaviourModel” as AutomationML “PLCOpenBehaviourModel”	50
Figure 29: Example AutomationML Representation of “PLCOpenBehaviourModel”	51
Figure 30: Example usage of the AutomationML role class “AMLLogic” and “BehaviourModel” as AutomationML “AMLBehaviourModel”	52
Figure 31: Example AutomationML Representation of “AMLBehaviourModel”	52
Figure 32: Example usage of the AutomationML role class “FMILogic” and “BehaviourModel” as AutomationML “FMIBehaviourModel”	52
Figure 33: Example AutomationML Representation of “FMIBehaviourModel”	53
Figure 34: Inheritance structure for an AutomationML “SimulationModel”	54
Figure 35: AutomationML Representation of “SimulationModel” role class	54
Figure 36: Inheritance structure of AutomationML “Sequence” role class	55
Figure 37: AutomationML representation of “Sequence” role class	56
Figure 38: Inheritance structure for an AutomationML “Function”	57
Figure 39: AutomationML Representation of “Function” role class	57
Figure 40: Inheritance structure for an AutomationML “SkillLogicModel”	58
Figure 41: AutomationML Representation of “SkillLogicModel”	58

Figure 42: Example usage of the AutomationML role class “GeometryModel”	59
Figure 43: AutomationML representation of “Model” role class	59
Figure 44: Example usage of AutomationML role classes to integrate an AutomationML “COLLADAGeometryModel”	61
Figure 45: AutomationML Representation of “COLLADAGeometryModel” role class	61
Figure 46: AutomationML Representation of “COLLADAGeometryAttachment” role class	61
Figure 47: Example usage of the AutomationML role class “JTGeometryModel”	62
Figure 48: AutomationML Representation of “JTGeometryModel” role class	62
Figure 49: Example usage of the AutomationML role class “2DGeometryModel”	63
Figure 50: AutomationML Representation of “2DGeometryModel” role class	63
Figure 51: Example usage of the AutomationML role class “Kinematic” role class	64
Figure 52: AutomationML Representation of “KinematicModel” role class	64
Figure 53: Example usage of AutomationML role classes to integrate an AutomationML “COLLADAKinematicModel”	66
Figure 54: AutomationML Representation of “COLLADAKinematicModel” role class	66
Figure 55: AutomationML Representation of “COLLADAKinematicJoint” role class	66
Figure 56: AutomationML Representation of “COLLADAKinematicAttachment” role class	66
Figure 57: Example of Connection between COLLADAKinematicJoint und FMILogic	68
Figure 58: Example usage of the AutomationML role class “GraphicalRepresentation”	69
Figure 59: AutomationML Representation of “GraphicalRepresentation” role class	70
Figure 60: Example usage of the AutomationML role class “Documentation”	70
Figure 61: AutomationML Representation of “Documentation” role class	70
Figure 62: Example usage of the AutomationML role class “Certificate”	71
Figure 63: AutomationML Representation of “Certificate” role class	71
Figure 64: Example usage of the AutomationML role class “AdditionalDeviceDescription”	72
Figure 65: AutomationML Representation “AdditionalDeviceDescription” role class	72
Figure 66: Example usage of the AutomationML role classes “MaintenanceDescriptionItem” and “MaintenanceDescriptionGroup”	73
Figure 67: AutomationML Representation of “MaintenanceDescription” role classes	73
Figure 68: General AutomationML Representation of Skill Description	74
Figure 69: AutomationML Representation of skill model role and interface classes	74
Figure 70: Role classes AutomationML Component connector definition	75
Figure 71: Interface classes for AutomationML Component connector definition	75
Figure 72: Representation of functions as AutomationML entities (UML diagram)	76
Figure 73: Example usage of an AutomationML “MechanicConnector” role class	77
Figure 74: AutomationML representation of “MechanicConnector” role class	77
Figure 75: Example usage of an AutomationML “LogicConnector” with one “SignalInterface”	77
Figure 76: AutomationML Representation of “LogicConnector” role class	77
Figure 77: Examples of electric connectors	78
Figure 78: Electric connectors used in a cable or in an automation devices	78
Figure 79: Connector versus Interface	79
Figure 80: Example usage of an AutomationML “ElectricConnector” with one ElectricInterface	79
Figure 81: AutomationML Representation of “ElectricConnector” role class	80
Figure 82: Example usage of an AutomationML “FluidicConnector”	81
Figure 83: AutomationML representation of “FluidicConnector” role class	81
Figure 84: Example usage of an AutomationML “LiquidicConnector”	81

Figure 85: AutomationML Representation of “LiquidicConnector” role class	82
Figure 86: Example usage of an AutomationML “PneumaticConnector”	82
Figure 87: AutomationML Representation of “PneumaticConnector” role class	83
Figure 88: AutomationML Representation of “PneumaticInterface” and “PneumaticConnector” interface class	83
Figure 89: Example usage of an AutomationML “HydraulicConnector”	83
Figure 90: AutomationML Representation of “HydraulicConnector” role class	83
Figure 91: Example usage of an AutomationML “SkillConnector” role class	84
Figure 92: AutomationML Representation of “SkillConnector” role class	84
Figure 93: Graphical visualization of the main contents of an AutomationML Component	85
Figure 94: CAD Model of a composite automation component and a graphical visualization of the belonging AutomationML Component	87
Figure 95: Inheritance structure of “AutomationMLComponentBaseRCL” and “AutomationMLComponentStandardRCL”	89
Figure 96: Overview “AutomationMLComponentBaseRCL”	90
Figure 97: Overview AutomationMLComponentStandardRCL as AutomationML tree	98
Figure 98: Overview AutomationMLFMLogicRoleClassLib as AutomationML tree	121
Figure 99: Overview AutomationMLComponentBaseCL	123
Figure 100: Overview AutomationMLFMInterfaceClassLib	129
Figure 101: Minimum AMLX Container for a single component	132
Figure 102: Minimum AMLX Container for a AutomationML Component catalogue.....	133
Figure 103: Minimum AMLX Container for an AutomationML Composite Component	134
Figure 104: UML Representation of role class “AutomationComponentSemanticSystem”	136
Figure 105: RefSemantic as AutomationML element	137
Figure 106: Examples for CAEX attribute “CorrespondingAttributePath”	138
Figure 107: Examples AutomationML Component with standard semantic system	139
Figure 108: Examples usage of semantic systems for attribute definition	139
Figure 109: Examples AutomationML Component with own semantic system	140
Figure 110: Examples usage of own semantic systems for attribute definition	140
Figure 111: Logic connector versus physical connector versus physical interface (pin)	142
Figure 112: AML interface class library modelling a subset of possible M12 interfaces according to IEC61076-2	143
Figure 113: General attributes of the abstract M12 class, inherited to every M12 variant	144
Figure 114: AML interface class library modelling a generic pin type	145
Figure 115: AML role class model of the M12 A coded with 4 pins and its male and female derivate	146
Figure 116: AML InterfaceClass library for Mini 7-8 interfaces.....	147
Figure 117: AML InterfaceClass library for an RJ45 interface	147
Figure 118: AML role class library modelling generic connector functions	148
Figure 119: AML SystemUnitClass of an AutomationComponent with two M12 Ethernet connectors	149
Figure 120: AML SystemUnitClass model of a single wire with two ends	150
Figure 121: AML SystemUnitClass model of a cable with 4 wires	150
Figure 122: AML SystemUnitClass of an M12 to M12 cable with 3 wires	150
Figure 123: Picture of the real pneumatic cylinder ADN-25-50-A-P-A that is modelled here as an AutomationML Component	151
Figure 124: An unpacked tree view of the contained file of the AutomationML Component.....	151

Figure 125: Object tree of the pneumatic cylinder with relations between models and connectors	152
Figure 126: Overview of the modelled connectors of the AutomationML Component	153
Figure 127: Contents of the AMLLogic behaviour model of the component	153
Figure 128: Visualization of the kinematic model of the component	154
Figure 129: Excerpt of attributes attached to the top-level element	154
Figure 130: Content of "ComponentPicture", "ComponentIcon", "PneumaticSymbol", "ManufacturerIcon" (from left to right)	154
Figure 131: Class diagram showing different aspects of skill of an automation component	155
Figure 132: VDMA SOArc and VDI 2860 based SkillConnector RoleClass	156
Figure 133: Automation component providing a skill	157
Figure 134: Example motor logic model	158
Figure 135: Example motor as internal element	159
Figure 136: Example motor definition of electrical connectors	159
Figure 137: Example motor integration of simulation model	160
Figure 138: Overview example frequency converter	161
Figure 139: Example frequency converter In-/Out interfaces	161
Figure 140: Example frequency converter connection to motor	162
Figure 141: Example integration AR APC	163
Figure 142: Example integration AR APC project structure	163
Figure 143: Example integration AR APC internal linking	164
Figure 144: AutomationML Editor SimulationModel references an external FMU (left) – pneumatic drive.fmu (right)	165
Figure 145: SIMIT InterfaceClassLibrary	166
Figure 146: SIMIT RoleClassLibrary	166
Figure 147: SIMIT Component Reference SINAMICS	166
Figure 148: SIMIT Component Motor Reference	167
Figure 149: SIMIT Component SINAMICS Reference	167
Figure 150: SIMIT Component SINAMICS Reference with PLC Signals	168
Figure 151: SIMIT Component Motor Reference	169
Figure 152: SIMIT Component Motor Reference with PLC Signals	169
Figure 153: Complete PLC Configuration with SIMIT as extension	170
Figure 154: CAD Drawing of a motor (red), adapter (blue) and linear positioning axis (grey)	171
Figure 155: SystemUnitClass describing a motor with its geometry and kinematic model	172
Figure 156: System unit class describing the motor-to-drive adapter with its geometry and kinematic mode	173
Figure 157: SystemUnitClass describing the drive with its geometry and kinematic model	174
Figure 158: InstanceHierarchy of drive train build of single components	175
Figure 159: AML role class library modelling M12 connector types according to IEC61076-2 ...	176
Figure 160: AML interface class library modelling a generic pin	177
Figure 161: AML role class model of the M12 A coded with 4 pins and its male and female derivate	177
Figure 162: AML role class library modelling communication protocols	177
Figure 163: AML SystemUnitClass of an AutomationComponent with two M12 Ethernet connectors	178
Figure 164: AML SystemUnitClass model of a single wire with two ends	178
Figure 165: AML SystemUnitClass model of a cable with 4 wires	178

Figure 166: AML SystemUnitClass of an M12 to M12 cable with 3 wires	179
---	-----

List of Tables

Table 1: Overview of AutomationML parts.....	19
Table 2: Overview of Stakeholders and Actors.....	25
Table 3: Examples for possible Connector to Model relations	86
Table 4: RoleClass AdditionalDeviceDescription.....	90
Table 5: RoleClass Connector	91
Table 6: RoleClass Documentation	91
Table 7: RoleClass GeometryModel.....	92
Table 8: RoleClass GraphicRepresentation	92
Table 9: RoleClass Icon.....	93
Table 10: RoleClass LogicModel	93
Table 11: RoleClass PLCopenXMLLogic	94
Table 12: RoleClass AMLLogic.....	95
Table 13: RoleClass FMLogic.....	95
Table 14: RoleClass KinematicModel.....	96
Table 15: RoleClass MaintenanceDescription.....	96
Table 16: RoleClass Model.....	97
Table 17: RoleClass Symbol.....	97
Table 18: RoleClass AutomationComponent.....	99
Table 19: Sub-Attributes "IdentificationData"	99
Table 20: Sub-Attributs "CommercialData"	100
Table 21: Sub-Attributs " ProductDetails"	101
Table 22: Sub-Attributs " ProductOrderDetails"	102
Table 23: Sub-Attributs " ProductPriceDetails"	102
Table 24: Sub-Attributs " ProductPrice"	103
Table 25: Sub-Attributs "ManufacturerDetails"	103
Table 26: RoleClass AutomationComponentSemanticSystem	104
Table 27: Sub-attributes of the "ClassificationSystem" attribute.....	105
Table 28: Attribute values "SemanticSystems"	105
Table 29: RoleClass BehaviourModel	106
Table 30: RoleClass Function.....	106
Table 31: RoleClass SimulationModel.....	106
Table 32: RoleClass SkillLogicModel	107
Table 33: RoleClass SequencingModel.....	107
Table 34: RoleClass Sequence	107
Table 35: RoleClass SequenceElement	108
Table 36: RoleClass COLLADAKinematicModel	108
Table 37: RoleClass COLLADAKinematicJoint	109
Table 38: RoleClass COLLADAKinematicAttachment	109
Table 39: RoleClass COLLADAGeometryModel	110
Table 40: RoleClass COLLADAGeometryAttachment	110
Table 41: RoleClass JTGeometryModel.....	111
Table 42: RoleClass 2DGeometryModel	111
Table 43: RoleClass ComponentPicture.....	112
Table 44: RoleClass ElectricSymbol.....	112
Table 45: RoleClass HydraulicSymbol	112

Table 46: RoleClass PneumaticSymbol	113
Table 47: RoleClass ManufactureIcon	113
Table 48: RoleClass ComponentIcon	113
Table 49: RoleClass SkillModel	114
Table 50: RoleClass Certificate	114
Table 51: RoleClass MechanicConnector	114
Table 52: RoleClass LogicConnector	115
Table 53: RoleClass ElectricConnector	115
Table 54: RoleClass FluidicConnector	115
Table 55: RoleClass LiquidicConnector	116
Table 56: RoleClass PneumaticConnector	116
Table 57: RoleClass HydraulicConnector	117
Table 58: RoleClass SensorConnector	117
Table 59: RoleClass SkillConnector	118
Table 60: RoleClass MaintenanceDescriptionGroup	118
Table 61: RoleClass MaintenanceDescriptionItem	119
Table 62: Values for Attribute "Cycle"	119
Table 63: Values for Attribute "ActivityKey"	120
Table 64: Values for Attribute "ExecutionKey"	120
Table 65: Values for Attribute "FunctionKey"	120
Table 66: Values for Attribute "PersonnelKey"	120
Table 67: RoleClass FMILogicObject	121
Table 68: InterfaceClass GraphicRepresentationReference	123
Table 69: InterfaceClass 2DReference	124
Table 70: InterfaceClass JTRreference	124
Table 71: InterfaceClass SkillInterface	124
Table 72: InterfaceClass DeviceDescriptionReference	125
Table 73: InterfaceClass MaintenanceDescriptionLink	125
Table 74: InterfaceClass MechanicInterface	125
Table 75: InterfaceClass ElectricInterface	126
Table 76: InterfaceClass LiquidicInterface	126
Table 77: InterfaceClass PneumaticInterface	126
Table 78: InterfaceClass PneumaticConnector	127
Table 79: InterfaceClass CondensateDrainConnector	127
Table 80: InterfaceClass HydraulicInterface	128
Table 81: InterfaceClass SensorInterface	128
Table 82: InterfaceClass JointInterface	128
Table 83: InterfaceClass FMIRreference	129
Table 84: InterfaceClass FMIVariableInterface	130
Table 85: Sub attributes for specifying the VDMA SOArc and VDI 2860 standards	156
Table 86: Description of connector types according to IEC61076-2	176

Contributor

Arndt Lüder, Otto-von-Guericke Universität Magdeburg

Sven Binder, Murrelektronik GmbH

Jörg Hinze, Murrelektronik GmbH

Joseph Briant, Schneider Electric

Mario Thron, ifak e.V.

Markus Rentschler, Balluff GmbH

Michael Dietz, Technische Hochschule Nürnberg

Michael John, Siemens AG

Rainer Drath, Hochschule Pforzheim

Hiroschi Yoshida, OMRON Corporation

Joachim Burlein, Daimler AG

Mathias Wiegand, Festo SE & Co. KG

Matthias Müller, Mitsubishi Electric Europe B.V.

Milan Vathoopan, fortiss

Glossary

Automation component

A component that supports partial or fully automated operation of industrial processes.

AutomationML Component Meta Model

The meta model of an AutomationML Component defines the model and structure of AutomationML Component. This includes the role classes and interfaces class that shall be used to describe type model and instance model of AutomationML Components, UML Diagrams defining the relations between different elements of an AutomationML Component and the specification.

AutomationML Component Type Model

A type model of an AutomationML Component or Composite Component describes a automation component or system. The root element of the Automation Component type Model shall be a SystemUnitClass. Within type models of AutomationML Composite Components all child lower AutomationML Components or Composite Components are modeled as InternalElements.

AutomationML Component Instance Model

AutomationML Component instance models of automation components are instances of AutomationML Components or Composite Components. The root element of instance models shall be an InternalElement. Instance models can be used within the InstanceHierarchies of an AutomationML file or as InternalElement within an AutomationML Composite Component.

AutomationML Component

An AutomationML Component is an AutomationML object that describes an automation component or system. This can be the type or the instance model of the automation component. Additionally, the automation component has no sub components. An AutomationML Component can be stored as SystemUnitClass or InternalElement.

AutomationML Composite Component

An AutomationML Composite Component is an AutomationML object that describes an automation component that is composed of AutomationML Components. An AutomationML Composite Component can be defined as type model or an instance model. An example for an AutomationML Composite Component is a handling system that combines two motors and a valve.

Automation System

A group of automation components interacting to support partial or fully automated operation of industrial processes.

Behaviour Model

A model that describes the responses or reaction of the component to sequencing information and/or to their external interactions.

Simulation Model

A behaviour model with focus on simulation. E.g. an executable software component that is capable to simulate a certain automation component's behaviour in interaction with other automation components. Usually it is run in a simulation environment with focus on evaluating the automation component regarding physics or time, e.g. simulation tool calculates the velocity profile of a moving axis pushing a heavy object vertically.

Sequencing Model

Sequencing model is information that describes the responses of the controlled system (or subsystem) to the sequencing information and to other external interactions. It is often represented by a given model which reacts on external input, e.g. behaviour of a gripper or valve. Those external inputs would trigger the behaviour or signaling of the gripper's states.

Skill

Potential of an automation component/system to provide a functionality required for the automated operation of industrial processes. Capability can be seen as a synonym for it. A skill can be made executable via a service interface and used as a connection point between different functional units that constitute an automated production system.

Technology DD

Device Description file according to the specification of a dedicated technology standard (i.e. a fieldbus).

1 Introduction

1.1 Overview

The engineering data exchange format AutomationML has traditionally aimed at connecting engineering software tools for planning and commissioning production plants. The focus of these engineering tools was on larger composites than components fulfilling functional roles in a production process.

Increased digitalisation and the trend towards multi-disciplinary, model based, and reuse driven engineering in the automation domain has had an impact on the engineering tool chains. For tasks like virtual commissioning, a higher granularity and an increased level of detail is needed to be modelled in the engineering process is needed.

In order to address the increased modelling demand on the automation component and system level, this whitepaper specifies methods and model structures that can be used to exchange component information as AutomationML Components in a way that allows the utilisation in complex, multi-disciplinary tool chains without tool specific post engineering efforts.

This whitepaper addresses different aspects of automation component and system description and is structured accordingly into the following sections.

Section 2 gives an overview of the concepts of AutomationML Components and Composite Components, the addressed aspects and use cases.

Section 3 describes the structuring of AutomationML Components, how to model single aspects of them and how to interlink single components.

Section 4 defines the standard libraries to describe AutomationML Components.

Section 5 describes a way how to exchange all information of an automation component or system in one AutomationML container.

Section 6 describes the extension of the AutomationML Component concept.

Section 7 gives a general recommendation for how to store semantic links for automation components.

Section 8 gives an overview how to describe AutomationML Components in CAEX 3.0

Section 9 is offering different examples how to use different aspects of the AutomationML Component description.

1.2 Basics

The data exchange format AutomationML is standardized in the IEC 62714 standard, which 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 parts¹

Part	Title	Description
Part 1 / WP Arch, V 2.0.0	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 / WP Lib V 2.0.0	Role class libraries	This part specifies additional AutomationML libraries.
Part 3 / WP Geo V 2.0.0	Geometry and kinematics	This part specifies the modelling of geometry and kinematics information.
Part 4 / WP Logic V 1.5.0	Logic	This part specifies the modelling of logics, sequencing, behaviour and control related information.
Whitepaper / WP Comm V 1.0.0	Communication	This Whitepaper describes the modelling of communication mechanisms in AutomationML
Whitepaper / WP eClass V 1.0.0	AutomationML and eCl@ss integration	This Whitepaper describes the integration of eCl@ss in AutomationML
Whitepaper / WP OPCUAAML V 1.0.0	OPC Unified Architecture Information Model for AutomationML	This Whitepaper describes a OPC UA Information Model to represent the AutomationML models
Application Recommendations/ AR APC V 1.2.0	Automation Project Configuration	This Application Recommendation describes a modelling method of automation project configuration data by means of the engineering data format AutomationML
Application Recommendations/ AR MES ERP V 1.1.0	Provisioning for MES and ERP – Support for IEC 62264 and B2MML	This Application Recommendation describes a method to link AutomationML objects to elements of IEC 62264.

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

¹ Additional Best Practice Recommendation to the standard can be found <https://www.automationml.org/o.red.c/publications.html>

1.3 Scope

This whitepaper proposes a structure and method for modelling automation components and automation systems in AutomationML. It is meant to be compatible with the documents on semantic referencing e.g. eCI@ss integration, communication, automation project configuration and all standardized parts of AutomationML. It focuses on the information needed for successfully handling automation components in a digitalised engineering process.

1.4 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.

- [WP-Part1:2018] Whitepaper AutomationML Part 1 - Architecture and general requirements, V 2.1.0, Jul 2018
- [WP-Part1:2016] Whitepaper AutomationML Part 1 - Architecture and general requirements Document Identifier, V 2.0.0 Apr 2016
- State: April 2016
- [WP-Part2:2014] Whitepaper AutomationML Part 2 - Role class libraries, V 2.0.0, Oct 2014
- [WP-Part3:2017] Whitepaper AutomationML Part 3 - Geometry and Kinematics, V 2.0.0, Jan 2017
- [WP-Part4:2018] Whitepaper AutomationML Part 4 - AutomationML Logic Description, V 1.5.0, Jan 2017
- [WP-eClassAML:2017] Whitepaper AutomationML and eCI@ss integration, V 1.0.1, Dec 2017
- [WP-Comm:2014] Whitepaper Communication, V 1.0.0, Sep 2014
- [WP-OPCUAAML:2016] Whitepaper OPC Unified Architecture Information Model for AutomationML, V 1.0.0 Mar 2016
- [BPR-EDRef:2016] BPR ExternalDataReference, V 1.0.0, Jul 2016
- [BPR-MLA:2016] BPR Modelling of List Attributes in AutomationML, V 1.0.0, Jan 2016
- [BPR-MlingExp:2017] BPR Multilingual Expressions in AutomationML, V 1.0.0, Mar 2017
- [BPR-DatVar:2017] BPR DataVariable, V 1.0.0, May 2017
- [BPR-Container:2017] BPR AutomationML Container, V 1.0.0, October 2017
- [BPR-Units:2017] BPR Reference Designation, V 1.0.0, September 2017
- [BPR-Units:2018] BPR Units in AutomationML, V 1.0.0, August 2018
- [BPR-RefDes:2017] BPR Reference Designation, V 1.0.0, September 2017
- [BPR-CLPAAML:2020] BPR CC-Link, V 1.0.0, June 2020
- [W3C1:2008] Extensible Markup Language (XML), W3C Recommendation, <https://www.w3.org/TR/xml/>, Nov. 2008
- [W3C2:2006] W3C: RFC 3986 - Uniform Resource Identifier (URI) Generic Syntax, <https://tools.ietf.org/html/rfc3986>, Jan. 2006
- [DIN77005-1:2018] DIN 77005 - Lebenslaufakte für technische Anlagen providing a standardized logical structuring of information on a production system component, 2018
- [RFC2046:1996] Multipurpose Internet Mail Extensions (MIME) standard described in RFC 2046, Nov. 1996
- [RFC5646:2009] Language Tags, RFC 5646, Sep. 2009
- [CI@ss:2020] eCI@ss classification and product description, Version 11.0, <https://www.eclasscontent.com/>, last visit February 2020

- [IEC 61360-4:2005] IEC 61360-4 Standard data element types with associated classification scheme for electric components - Part 4: IEC reference collection of standard data element types and component classes, 2005
- [IEC62683-SC3D:2014] IEC 62683 - SC 3D / SC 17B - Common Data Dictionary (CDD - V2.0014.0016), 2014
- [IEC61131-10:2019] IEC 61131-10 Programmable controllers - Part 10: PLC open XML exchange format, V 1.0, Apr 2019
- [IEC 61987:2016] IEC 61987-11:2016 Industrial-process measurement and control - Data structures and elements in process equipment catalogues - Part 11: List of properties (LOPs) of measuring equipment for electronic data exchange - Generic structures, 2016
- [ISO 14306:2017] Industrial automation systems and integration - JT file format specification for 3D visualization, Edition 2, Nov 2011
- [FMI:2019] Functional Mock-up Interface for Model Exchange and Co-Simulation, (available at <https://fmi-standard.org/downloads/>), Version 2.0.1, Oct 2019
- [OPCUA-Part100:2020] OPC UA Part 100 - Device Information Model, Version 1.02.02., June 2020
- [ISO/IEC 29500-2:2012] Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 2: Open Packaging Conventions, 2012
- [ISO 18582-2] ISO 18582-2:2018-11, Fluid power - Specification of reference dictionary, Part 2: Definitions of classes and properties of pneumatics, 2018

2 Automation Component

A mechatronic production system component (consisting of automation components) interacts with other automation components within a complex production system in different ways. The design of production systems involves different engineering disciplines covering -but not limited to- functional, mechanical, electrical, control and HMI engineering. Each discipline needs specific information aspects about an automation component. The approach of this whitepaper is to present a meta model for an AutomationML Component that allows to describe such automation components and production systems with all these aspects on different levels. Within these meta model, functional roles are applied to identify information related to aspects by using role classes within AutomationML. Figure 1 gives an overview about a selection of different aspects that belong to an automation component or production system.

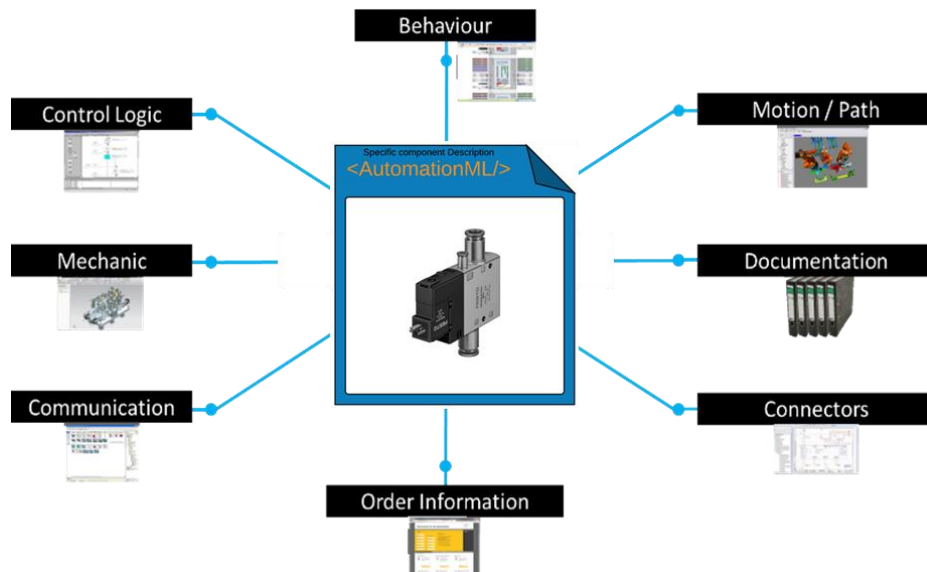


Figure 1: Aspects of an automation component or production system

Another aspect that is considered within the AutomationML Components is the relations the automation component can be within. To model this, it is necessary to describe interfaces and connectors of automation components to interlink them and to define internal linking of component aspects and external information references.

The lifecycle of production systems encompasses various phases that can benefit from the use of consistent component descriptions by providing consistent information to the applied tool chains.

Tools that are involved in this process may be:

- Plant planning tools
- Mechanical engineering tools (MCAD)
- Electrical engineering tools (ECAD)
- PLC programming tools
- Robot programming tools
- HMI programming tools
- OPC UA system configuration tools
- Device configuration tools
- Bus configuration tools
- Simulation tools

- SCADA systems
- Virtual commissioning tools
- Documentation tools
- Communication system security tools
- Communication system configuration tools
- Communication system management tools
- Communication system diagnosis tools

Figure 2 depicts a set of these tools using AutomationML Component information within the development process.

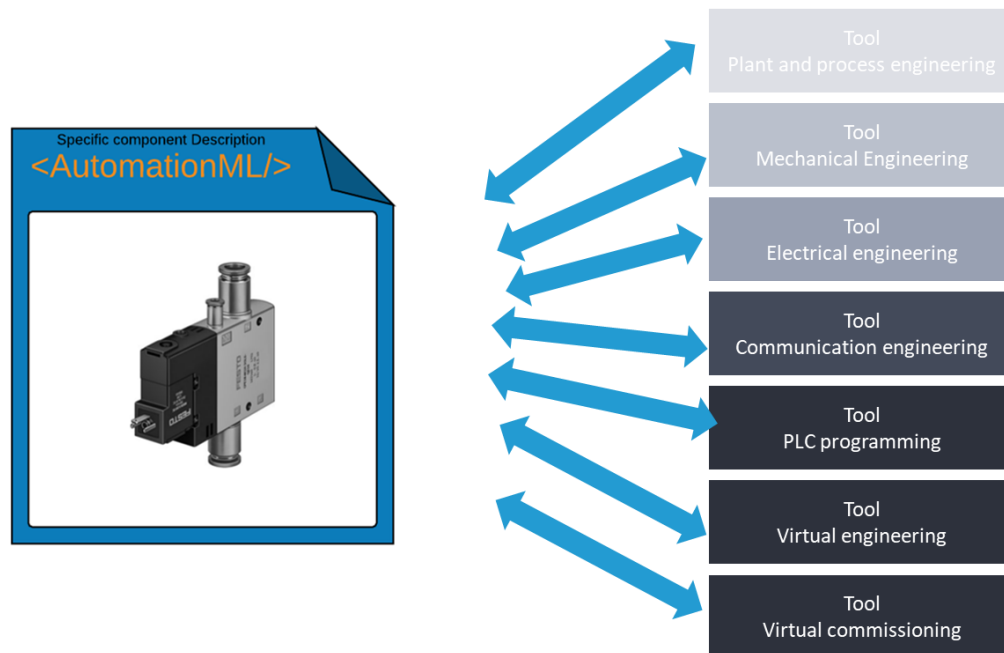


Figure 2: Overview of tool use for AutomationML Components

2.1 Stakeholders and Lifecycle

The model of an automation component and production system described as AutomationML Component has in general three stakeholder groups during its lifecycle.

- The first stakeholder group is a Standardisation Consortium, here the AutomationML Association, who defines the meta model of AutomationML Components.
- Second stakeholder group encompasses the Component Manufacturers (Vendors), who create their type model AutomationML Component libraries based on the meta model from the Standardisation Organisation. In this way the meta model serves to ensure interoperability between different manufacturers and consistency of the models throughout its life cycle.
- This is assumed by the third stakeholder group, the Component Users e.g. System Integrator or OEMs, who uses instances of the component description, the instance models, within their engineering processes. The Component User can do the conception and engineering of the system by interconnecting the interfaces of individual instance models of AutomationML Components, which results in an AutomationML model of the system. Without in-depth knowledge, the domain related engineering data can be generated automatically from the AML system model using generative programming. Any reconfiguration data of the domain related tools can be exported back to the AutomationML system model. The End User, can use the component models also for operation and maintenance.

Figure 3 depicts an overview about the stakeholder groups and the used models of an AutomationML Component.

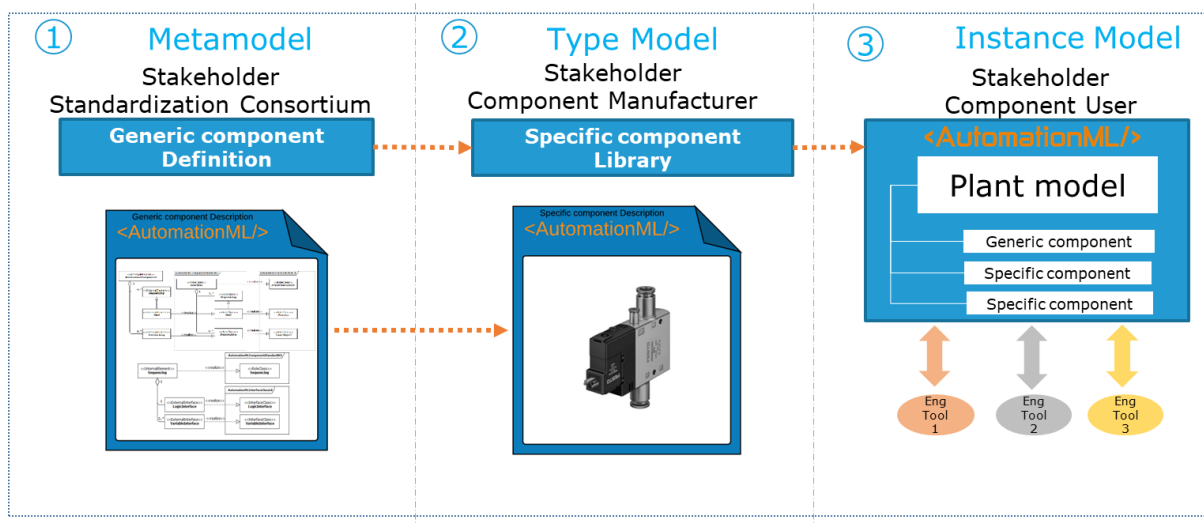


Figure 3: Stakeholder of automation component

Within the context of the AutomationML Component description the meta model, the type model and the instance model include the following AutomationML model element.






- **AutomationML Component Meta Model**
The meta model includes all role classes and interface classes that shall be used to describe a AutomationML Component or AutomationML Composite Component as type or instance model. It additionally includes normative restrictions of the use of these elements and UML diagrams to define the cardinalities between them. The meta model of AutomationML Components is defined within this whitepaper.
- **AutomationML Component Type Model**
The type model of an AutomationML Component or Composite Component is the generic description of an automation component type as system unit class. Usually a type model is defined by the vendor of the automation component and is stored and distributed within a system unit class library.
- **AutomationML Component Instance model**
The instance model of an AutomationML Component or Composite Component is the representation of an instance of an automation component within an AutomationML instance hierarchy as AutomationML object.

The lifecycle of production systems consists of various phases. All these phases can benefit from the use of consistent component descriptions by providing consistent information to the applied tool chains.

In the system development phase of the production system for example, the overall resource structure, mechanics, electronical structure and software of the system are designed. Tools for plant and process planning, CAD tools (including MCAD and ECAD), programming tools and configuration tools (especially for communication systems) are utilized during this phase. In parallel to the design, system validation activities are carried out. These validation activities can be divided into Virtual Engineering (VE) and Virtual Commissioning (VC).

Table 2 gives an overview about the stakeholder groups, the stakeholder within these groups and possible actors that can use the different AutomationML Component models during the lifecycle of a production system.

Table 2: Overview of Stakeholders and Actors

Stakeholder	Symbol (provided by Free Industry Illustration)	Description	Actors
Stakeholder Group 1: Standardization Consortium			
AutomationML Association	<AutomationML/>	Provides Meta Model	Association Members
Stakeholder Group 2: Component Manufacturer			
Vendor		Provides products, such as devices or (sub-)systems to the System Integrator, including firmware	Manufacturer, OEM
Stakeholder Group 3: Component User			
Application Supplier		Provides Services and Tools that use the functionalities of the devices or (sub-)system.	Application Product Manager, Application Developer, Application User
System Integrator		Integrates components, services and tools to (sub-)systems, e.g. devices with higher-level systems, to entire solutions Sets HW components into operation, including configuration	Plant Planner, Machine Builder, Solution Provider, Mechanical Engineer, Electrical Engineer, Software Engineer, PLC Engineer, Project Manager, MES/Process Engineer, Application Engineer, Commissioning Engineer
Operator		Uses the facility by running normal operations	Operator, Facility Owner, End User, Application engineer
Maintenance Staff		Maintains facility, including exchange of components, firmware updates	Maintenance engineer, Application engineer

In Figure 4, a typical engineering and operational data flow between stakeholders in the automation component or production system lifecycle is indicated. Within this dataflow, the manufacturer of the component delivers a type model of its automation component to the system integrator. The system integrator specifies the instance model of the automation component as AutomationML Component and

uses it during the engineering and commissioning phase. Additionally, the AutomationML representation of the automation component or system can be used during the start up and operation phases for additional use cases.

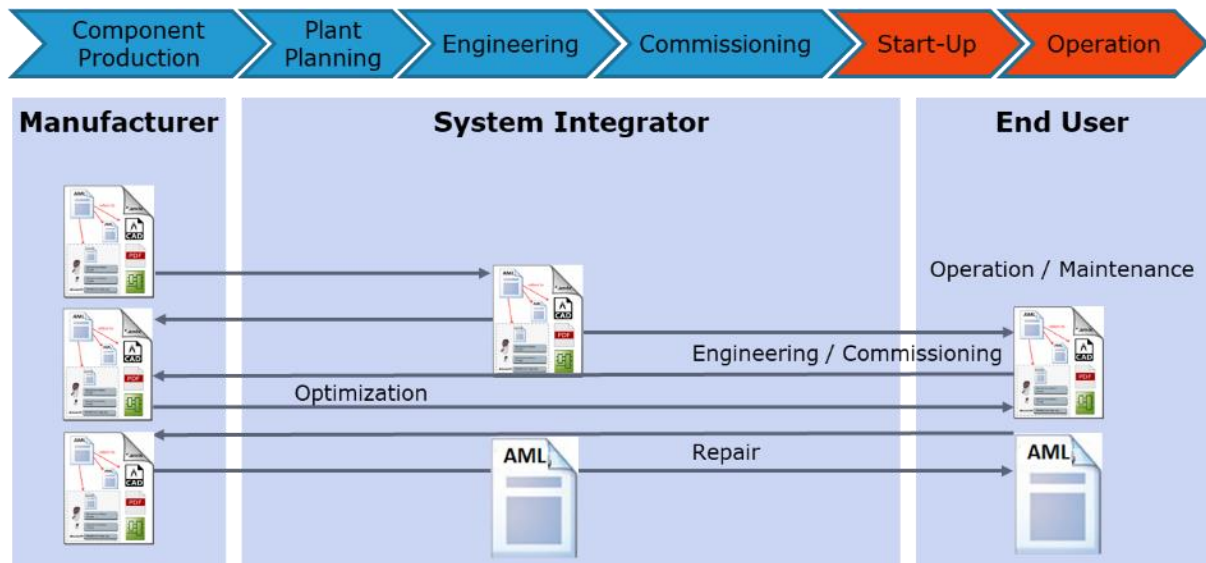


Figure 4: Data flow between stakeholders over the lifecycle

2.2 Use Cases

Component related information is relevant in various engineering activities along the engineering chain of production systems. Within the engineering process of production systems, automation components can be defined in the single engineering phases exploiting various tools. Thereby, automation component information is created. This should subsequently be applied within the detailed design of devices and the device commissioning. Within the different engineering activities, specialized engineering tools are used and will have a relevant impact on the automation component information.

These tools will create and/or consume component related engineering information depending on the use case within the engineering chain.

Within the following sub subchapters some relevant used cases for the exchange of automation components will be introduced.

2.2.1 Materials Management and Warehousing

One of the results of an engineering workflow is a bill of material, that is used to order the needed spare and wear parts for the stores. A proper identification of the parts is needed. Therefore, part numbers or product code order number shall be exist to find existing parts.

Some companies prepare material release lists for bigger projects, to standardize the used parts. This is a way to avoid uncontrolled growing of stores. Therefore, the standardized catalogue information is necessary, including all relevant documents for the planning process, engineering process, maintenance, commissioning and decommissioning. In an advanced engineering scenario, all participating engineering tools can reference the same part, this is needed for an efficient data processing at the end of an engineering process.

2.2.2 Component Description as Base for xCAD

This use case describes three examples of how data of the AutomationML Component description provided in AutomationML files by Component Manufacturers can be used in engineering tools of Component Users. Figure 5 depicts the data flow from two different AutomationML Components from two Component Manufacturers to the engineering tools of a Component User (import). Out of this data

and further engineering information these tools build a AutomationML system that virtually represents the real automation system. The resulting engineering data can be a rich input for virtual commissioning.

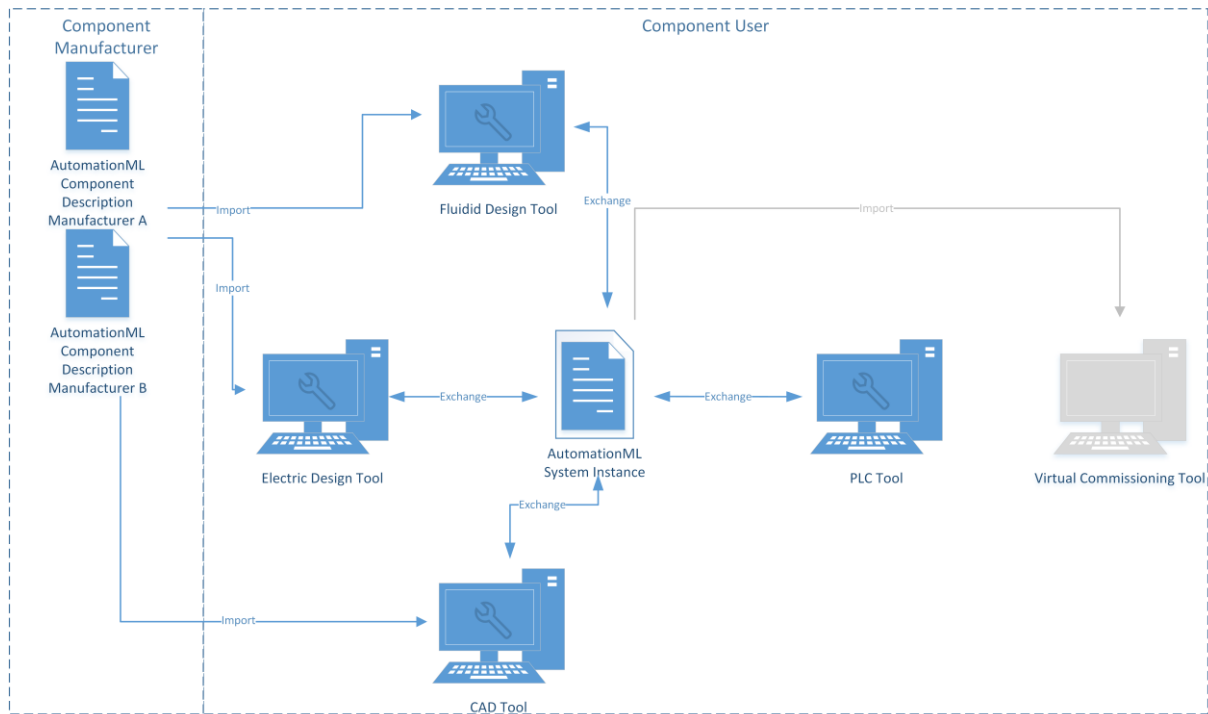


Figure 5: Use Case component description a base for xCAD

The following three use cases describe the benefits of AutomationML Components in more detail.

2.2.2.1 Electrical Engineering and software design

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 view of the automation devices is used. In ECAD tools there are defined devices which are involved in an automation system e.g., power connectors that are used for the power supply of the devices and wire types which are used to connect devices logically. But this information is not used in PLC tools, where device and control application specific properties are used, e.g. baud rates within the communication connections, control code variables that are associated to control device inputs and outputs, and control application codes. This kind of information is 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 to access the device within the PLC.

Recommended Workflow

In a production system engineering process, the construction phase in the PLC project usually begins later than in the ECAD project, because the completion of the ECAD documents is the base for the production of the control cabinet. The combination with the PLC software within the plant and the following commissioning will not take place before all control cabinets are completed. So the PLC engineer will usually start his project work later than the electrical engineer. Nevertheless, at an early point of time (during ECAD engineering), the components of the plant must be defined because the ECAD documents must be generated and the parts must be ordered.

ECAD systems normally can handle the components from a point of view of electrical hardware of different PLC manufacturers, because they have certain analogies. For this, the components must be defined in a neutral model. According to the described criteria, for most cases the following workflow can be established.

- Engineering of the basic device configuration within the PLC project of the PLC programming tool and exporting it to ECAD tool:
- If no ECAD project exists so far, the electrical engineer first defines a raw project within the engineering system of the PLC manufacturer, the PLC programming tool. The electrical engineer selects all needed components and defines the fieldbus or network 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.
- Importing PLC project to ECAD tool, engineering of the ECAD project, and exporting the ECAD project to PLC programming tool:
- Based on the existing ECAD project, the electrical 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
- Importing ECAD project into PLC programming tool and engineering of the PLC project: Now the ECAD project can be exported to AutomationML dependent from the implementation in the ECAD tool and imported to the PLC programming tool. Now the PLC programmer will begin the engineering based on the already developed ECAD project. So, at this point of the engineering process all components shall be defined regarding electrical and logical connections.

2.2.2.2 Mechanical Engineering

In the mechanical engineering of a machine it is necessary to combine various components of different vendors. To be able to do that a lot of information regarding these components is necessary. Some of this information can be found in the CAD model of the component. But usually it is necessary to consult additional documents such as manuals, catalogues, etc. to gather all the required information.

Assumed/Recommended Workflow:

The mechanical engineer creates the design of the machine. When the rough planning is finished it is necessary to select the factory automation components that are required to fulfill the requirements of the machine. E.g. a motor needs to be selected to move a ball screw.

In this case the mechanical engineer receives the component description from the FA component supplier. This description contains all aspects of the component which is needed to add it to the mechanical design. This includes the CAD data itself but additionally also parameters for the components. It also includes a description of the mechanical connectors of the component. With this information it is easily possible for the mechanical engineer to add this component into his design. He can check whether the components fit to the machine, whether the mechanical connector fits to the connector at the machine side, he can decide what kind of screw etc. he needs to attach the component to the machine.

Example

The mechanical engineer wants to use a ball screw from Vendor A together with a servo motor from Vendor B.

The AutomationML Component Description of the ball screw tells that there is a mechanical connector for the motor shaft and its precise location in the CAD model. It also tells the certain type of screws required to fix the motor.

The AutomationML Component Description of the servo motor tells also that there is the motor shaft as a mechanical connector and that there is the mechanical connector for the screws.

With this description it is easily possible to check whether this motor fits to this ball screw and it can be integrated into the mechanical design.

2.2.2.3 Fluidic Engineering

In the engineering of fluidic systems, components are put together by logically and physically interconnecting them. There are several tools that can handle fluidic representations of components but there is no standard way of importing fluidic component symbols or even logics into those tools. The lack of exchanging fluidic system with tools from other domains, for example, ECAD or PLC is a burden in efficient engineering.

Assumed/Recommended Workflow:

When engineering is at the stage of the fluidic construction the fluidic representation, e.g. the fluidic symbols of the chosen components can be imported in the tool for fluidics. It doesn't matter from which vendor the components are, because the file format is standardized and the symbolics just as well.

If the fluidic tool supports simulation of the fluidic network, it benefits from the logic behavior that is described in the component file too.

Example:

Each fluidic vendor provides a component description for his components. The machine builder can download it and import it to the tools of his choice. He might make use of the additional information and models inside the integrated component description in order to do some virtual functional tests and timing calculations. After creating the fluidic network and applying changes to components he can save the file and exchange it with other tools of the engineering process, for example with the mechanical design tool, ECAD, PLC or virtual commissioning tool.

2.2.3 Simulation and Virtual Commissioning

Principle of Virtual Commissioning

The purpose of Virtual Commissioning (VC) is to validate the control code by use of a simulated process environment instead of the real physical machines or plants. Thus, the overall system behavior can be tested in virtual standard and extreme situations before deploying the control code to the controller hardware and thus multiple tests are necessary.

A single test is a mapping of a test stimulus to an expected behavior of the process environment. The test stimulus defines the start situation of the virtual environment and control system and a set of triggers, that lead to changes in the simulated process environment. The expected behavior is defined by process parameters, that have to be in a certain range after a specified time. An automated test system is able to perform a variety of tests and to report about deviations of the observed simulated system behavior from the expected one that indicate possible problems in the control code.

In VC scenarios, the control code can be interpreted by simulation blocks (software-in-the-loop, SIL) or it can run directly on the target hardware (hardware-in-the-loop, HIL). In case of HIL, the controller is represented by a proxy block within the overall simulation environment. That control proxy exchanges all necessary signals between the simulation environment and the physical controller behind the scenes. Figure 6 provides an overview of a simulation environment by use of the HIL pattern, while the controller is represented by a control proxy block.

The simulation model is expected to be of modular nature, since it enables reuse of model parts that represent components, which is essential for efficient VC of complex systems. It is a practical approach to provide a simulation model with a corresponding mechatronic component, since it is related to the type of the equipment. The developer of a simulation model for a component needs deep insight into the behavior of the component. Thus, the manufacturer of the component becomes also a candidate for the development of the corresponding simulation model.

One of the main questions is how to develop the component simulation models. AutomationML whitepaper part 4 introduces languages like PLCopen XML to describe the behavior of components.

The usage of a harmonized language, that describes the component behavior as open source often provides insight into the construction of the component itself. This provides not only the benefits to the operator of the production system, but it also provides access to construction details that the manufacturer of the component wanted to be kept secret with regard to market competitors. To protect this kind of intellectual property, an alternative model language based on the public available simulation interface standard Functional Mockup Interface (FMI, version 2.0) can be used, which is adopted by several simulation system vendors. Especially the co-simulation variant is selected, which enables a better hiding of models and simpler co-simulation masters compared to the model exchange variant of FMI. FMI for co-simulation defines Functional Mockup Units (FMUs) in form of a ZIP file containing an XML-based interface description (modelDescription.xml) and one or more binaries of simulation files as shared libraries for a variety of operating systems (Windows, Mac, Linux in 32-bit or 64-bit variants).

Controllers can be represented by proxy FMUs that communicate in background via fast communication interfaces with the controller hardware as depicted by Figure 6.

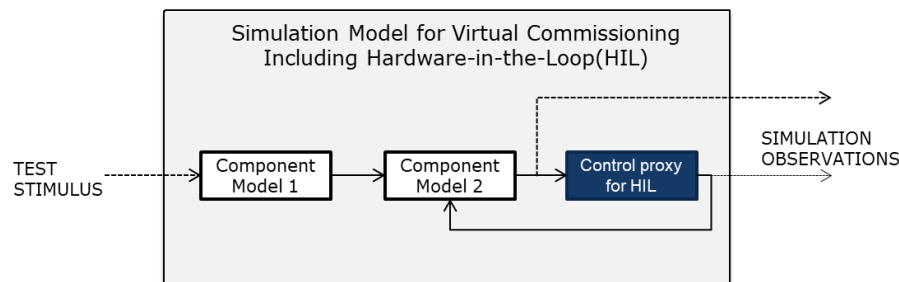


Figure 6: Schematic example of an FMI-based co-simulation model for VC including HIL

The use-case for VC by use of AutomationML-based information could be as follows:

- 1) Design the system by using xCAD systems that maintain the component structure and export this structure as AutomationML.
- 2) Assign FMUs to the components and store that information in an enhanced AutomationML Component container.
- 3) Interconnect the FMUs and store the interconnection information in a further enhanced AutomationML container.
- 4) Use the result AutomationML file to configure the co-simulation master and define further simulation parameters.
- 5) Configure the Stimulus Generator and Comparator.
- 6) Start the simulation.
- 7) Evaluate the test reports.
- 8) If there are reported problems, then adapt the component structure and/or the controller programs accordingly (involves partially steps 1 to 4) and go to step 6. If there are no reported problems, then the VC has been finished successfully.

Steps 2 and 3 contain tasks, which are related to the creation of AutomationML in context of VC based on FMI for co-simulation. Thus, we will clarify how to represent FMUs, interconnections between FMUs and how to address inputs and outputs of FMUs in chapter 3.5.5.

2.2.4 Maintenance and Documentation

After having the production system established it is utilized within the value creation process. This utilization results in wear and tear of the production system components. Therefore, production system maintenance is required enabling the system user to react proactively on potential or occurred system failures.

Independent from the applied maintenance strategy, prerequisite for successful and efficient maintenance is the availability of a detailed and up-to-date system documentation covering all relevant details of the system design and utilization state. This must all be considered when defining component descriptions.

Currently the necessary maintenance tasks during the production phase are mostly created by hand. Technical editors of the system integrator have to inspect all manuals of the used parts in a production system to list all necessary tasks for preventive maintenance. This data is going to be copied to the manual of the production system. Maintenance planners from the customer are inspecting the production system manual to get all tasks for preventive maintenance. Almost the same process takes place for technical documentation.

The optimal workflow for preventive maintenance tasks is to get the tasks in a standardized way, in our case inside an automation component.

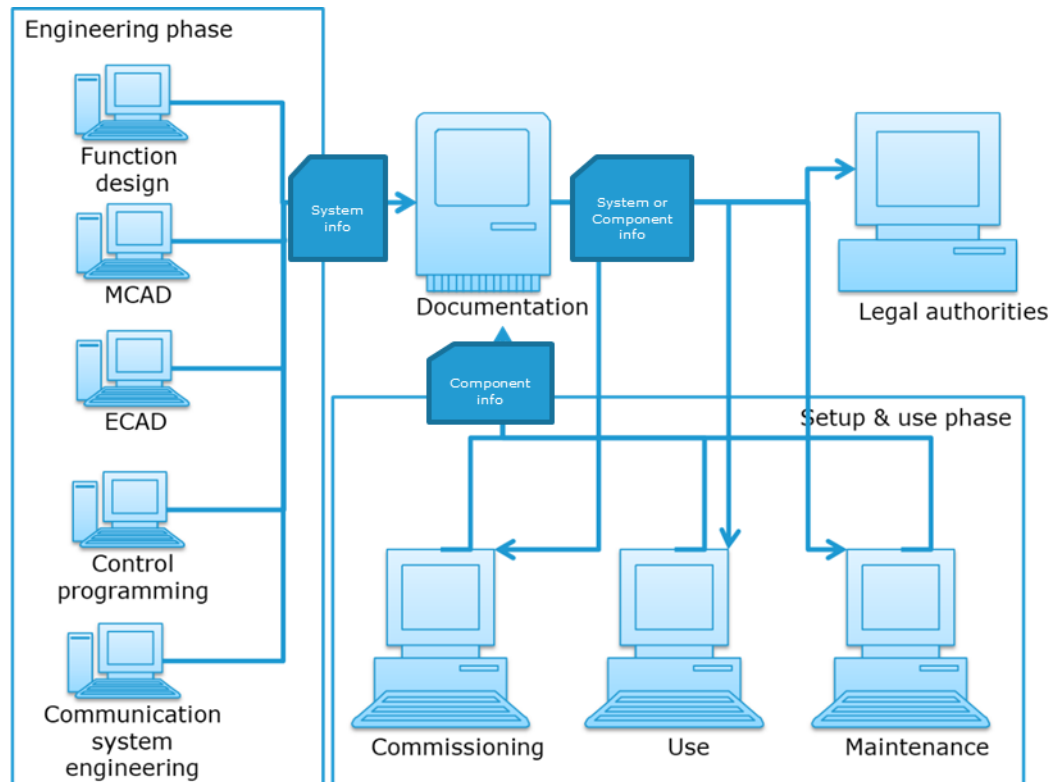


Figure 7: Use Case Maintenance and documentation

Assumed/Recommended Workflow

The production system documentation process is usually started at the end of the engineering phase, summarizing all relevant engineering information covering -but not limited to- function design, mechanical engineering, electrical engineering, communication system engineering and control programming. Therefore, the related engineering tools have to provide their results to the documentation tool.

The documentation process is continued during production system commissioning by updating engineering results following the changes made during the physical realization of the production system and integrating information on the labelling of system components (like wire marking or printed reference signs). Therefore, the related commissioning tools have to provide their results to the documentation tool.

The documentation process is also continued during the production system use phase. Here replacements of production system components and their impact in the mechanical, electrical, control, etc. construction of the production system shall be documented to ensure an as-is documentation. Therefore, the related engineering, commissioning, or maintenance tools have to provide their results to the documentation tool.

In addition, during the operational phase, KPIs representing the production system component utilization can be collected. Therefore, the appropriately designed control applications need to provide the relevant KPIs or their basic data to the documentation tool.

The documentation process ends after the disposal of the production system.

The created documentation can be exploited by maintenance tools to support maintenance staff in two directions. In predictive maintenance scenarios the collected KPIs and the available as-is construction data of the system are exploited to define maintenance schedules and to define maintenance procedures. In reactive and predictive maintenance scenarios the available engineering information can be exploited to support the maintenance staff within planning and execution of the relevant maintenance activities. Therefore, production system component related KPI and as-is engineering and identification data need to be exchanged between documentation tools and maintenance tools.

2.2.5 Device Description Files for Field Devices

Modern field devices for process and factory automation often have a number of identification and configuration options and are customizable to their individual use case throughout the plant lifecycle. For this purpose, they are equipped with a digital communication interface, such as IO-Link, HART, PROFIBUS, Fieldbus Foundation, CC-Link, Ethernet/IP, PROFINET etc. Each of these communication standards has developed its own dedicated software tool ecosystem to control and configure the devices, usually based on a *Device Description Language* (DDL) approach, allowing generic software tools to configure and control different devices through the interpretation of a *Device Description* (DD) associated to the individual device type. The economic benefit of this approach is based on the fact that the creation of a DD with the DDL requires much less effort than writing a dedicated software tool for each device type.

The newer XML-based formats such as GSDML, FDCML, ESI, CSP+, IODD and others offer advantages compared to the traditional text-based formats GSD, EDDL and EDS, because they can rely on data model schematics (XSDs) and the related XML parser features for consistency checking of both syntax and semantics.

The DD files are supplied by the device vendor for each device type. The DDs are loaded and interpreted in the engineering software tool, providing user dialogs and functionality to enter property parameters in order to configure the individual device instance. All devices of the same type have the same DD file, but the individual parameters of individual devices may have different parameter values dependent on the use case of a device. These “Technology DD” suffer from two following general limitations:

- In many cases there is a strict split between type information and individual device information. This implies that type specific information is stored in the DD file, while the individual parameters are stored in the proprietary engineering tool. No tool independent storage of individual device configuration across the devices life cycle is established.
- The classic approach of device descriptions defined by fieldbus organizations is usually focused on just modelling the primary fieldbus interface of a device and lack of modelling capabilities for many other important aspects of a device, such as secondary interfaces, power sourcing interfaces, functional models and mechanical models.

AutomationML provides mechanisms to reference third party files, which enables referencing these files from within AML files, forming an AutomationML Component as Device Description. This would allow to keep the standard third-party file as it is, but to model additional information on top of it within the AutomationML part. The idea is therefore to add an AutomationML model on top of the respective third-party models. Whereas the third party standards usually delivers only the e.g. fieldbus specific type information like Device Descriptions of a device, AutomationML can provide additional model descriptions, classes and also instances with individual configurations, plus the modelling of hierarchies and links between object instances, which allows also the representation of modular devices.

The usage of AutomationML will enable the device and tool vendors to overcome the present limitations in a standardized way. The device properties shall be based on the native device description information that can be enriched with advanced modelling information as defined in this document. In the following it is intended to define a standardized procedure for this approach. It is envisioned that these future

AutomationML based Device Description can also form the building blocks of larger automation system models for heterogeneous technologies.

To gain acceptance for this approach within the automation industry, also a generic and simple migration path from current fieldbus DDs shall be defined and described to protect the stakeholder's tool investments.

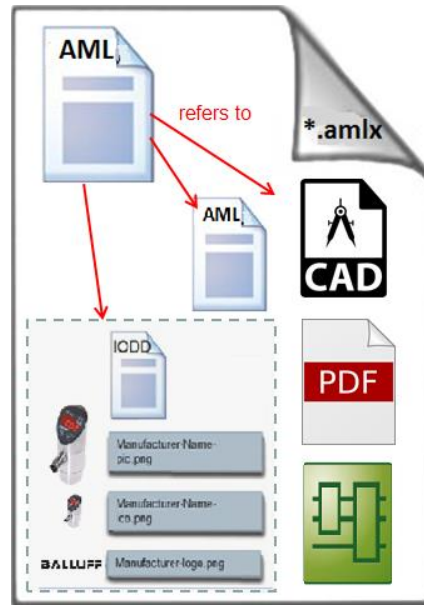


Figure 8: Principle of the AML-DD package with an example for an IO-Link device

Recommended basis for the packaging method of the AutomationML based Device Description is via AMLX Container, described in [BPR-Container:2017] and furthermore a detail description of packing AutomationML Components via AMLX Container can be found in chapter 5.

2.3 Considered Information and sub-model of Automation Components

Within the scope of the use cases named above, different information and partial models of an automation component must be mapped or integrated in the AutomationML Component.

This is done using two mechanisms. Firstly, information about the automation component is stored as attributes of the component and secondly, sub-models of the component are modelled as InternalElements of the AutomationML Component or linked via these

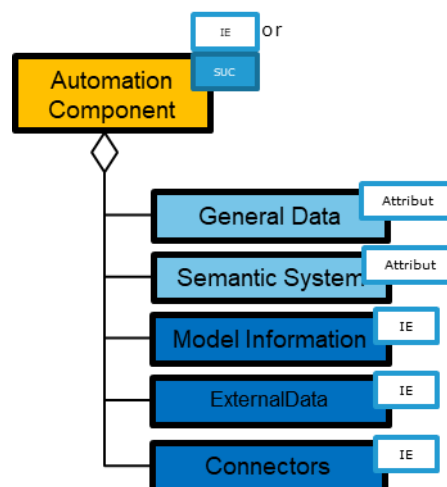


Figure 9: Information and sub-model integration into the AutomationML Component

Figure 9 shows schematically how information and sub-models are integrated into the AutomationML Component.

In general, the following categories can be distinguished.

1. General Data

This part incorporates component identification and classification data from the vendor, like the product identification code, ordering information, serial number, version number, other technical information. All this information is mapped to attributes of the root of an AutomationML Component model.

2. External Data

This includes for example the documentation of the automation component and symbols and pictures of it. The external data are linked to the AutomationML Component via an ExternalInterfaces that belong to a child InternalElement of the AutomationML Component root.

Note 1: It is highly recommended to align with specifications like VDI 2770 and the "VendorNameplate" concept of OPC UA Device Integration companion specification [OPCUA-Part100:2020], including semantic references for example to IEC 62683-1.

3. Model Information

The model information comprises different kinds of models representing the automation component. Examples for such models are.

- Functional Data
- Simulation models
- 2D and 3D models
- Kinematic models

These Models are linked as well to the AutomationML Component via an ExternalInterfaces that belong to a child InternalElement of the AutomationML Component root as the external data.

4. Connectors

The connectors describe information to all logic, electric, pneumatic, hydraulic and other interfaces of the component. These connectors are modelled as ExternalInterfaces that belong to an InternalElement of the AutomationML Component.

2.4 Views on Automation Components and Systems

The engineering and operational processes of automation components and systems are multi-disciplinary. As a consequence, also the information related to automation components and systems components are multi-disciplinary.

Therefore, there are various views on the system component related information depending on the considered life cycle phase, activity or action. It is impossible to identify all possibly relevant views on production system component data. Therefore, in the following, some main views and their relevance for the AutomationML Component meta-model are sketched.

The general life cycle phases of a production system are establishing main views on production system component data. These are the following:

- The engineering view summarizes all engineering related information including function design, mechanical engineering, electrical engineering, communication system engineering, and control programming defining the as-designed be status of the production system including the virtual representation of its expected status handled within virtual engineering and virtual commissioning.
- The commissioning view summarizes the as-built information of the production system including the function, mechanical, electrical, communication system, and control programming realization details.
- The operations related view summarizes the as-is information of the production system including the function, mechanical, electrical, communication system, and control programming state details

including possible virtual representations of the production system components as well as documented KPI histories.

Each of these general views can be divided in more specialized views following the life cycle activity considered. There are for example (without being limited to):

- Engineering related views on the production system are established by (for example):
 - production system function architecture defining the component hierarchy,
 - mechanical construction defining the mechanical structure of the production system including for example mechanical joints between components,
 - electrical construction defining the electrical structure of the production system including for example the wiring,
 - control programming defining the complete data processing infrastructure including all relevant control applications on the different level of control,
 - communication system engineering defining the complete communication system including fieldbus configurations.
- Commissioning related views on the production system are established by the documentation and its substructures.
- Operations and maintenance related views on the production system are established by (for example):
 - Production system KPIs for system utilization on manufacturing execution level,
 - Production system KPIs for system utilization on component wear and tear out level,
 - Production system maintenance actions.

These named more technical oriented views are summarizing all information relevant for the complete discipline. But there are also very specialized views on production systems following the special needs of an individual action within the production system life cycle.

Examples of such very specialized views (without claiming completeness) are:

- Drive list required for the engineering (dimensioning) of power supply systems,
- Multibody simulation model required for validation of the mechanical behaviour of system components within virtual engineering,
- Wring list required to document the planned clamping positions of individual wires for commissioning,
- Utilization status list representing the number of use actions required for maintenance decisions.

All views have a common structure. They cover system components with their properties and relations.

In case of automation components as special components of production systems these views are designated to the engineering and utilization of automation structures. Therefore, they cover but are not limited to the following engineering activity related views:

- Planning of the static structure of the production system requiring the spatial and hierarchical relations among the components regarding their physical measures.
- Planning of physical interconnections between automation components requiring the representation of necessary physical connection points of automation components including their properties regarding the physical measures themselves, or alternatively by referencing connector standards.
- Planning of the functional behaviour of the automation system requiring the description of provided functionalities, connection facilities and life cycle management rules of components within human readable manuals.
- Simulation of the interaction between automation components and their controlled objects (mechatronical objects) to enable an efficient design and commissioning of complex plants requiring behaviour models including component and dependency behaviour models.

Manufacturers of components therefore should bundle manuals, physical connector information, physical simulation models as well as 2D and 3D models within their component documentation. AutomationML InstanceElements may be used to represent those components.

Additional views are defined and discussed on [DIN77005-1:2018] providing a standardized logical structuring of information on a production system component.

The definition of specific views on AutomationML Components for different engineering disciplines will be part of the further work of the AutomationML team.

2.5 AutomationML Base Technology CAEX 2.15 and CAEX 3.0

CAEX 2.15 is the basis for AutomationML Version 2.0, while CAEX 3.0 is the basis for AutomationML Version 2.1. Currently there are no standardized libraries for version 2.1. The libraries already developed refer to AutomationML Part 1 [WP-Part1:2018] and have not yet been released. Libraries of the other parts are not yet defined. The libraries defined here for modeling components use basic libraries of all previously published AutomationML parts. These libraries must therefore be created on the basis of the existing AutomationML Version 2.0 libraries.

A pure schema conversion of the CAEX 2.15 version of the libraries to the CAEX 3.0 schema can be done later without problems. For the transformation of the AutomationML version, which provides for a transformation of the libraries to new versions of the standard libraries, there is as yet no software solution that carries out the transformation automatically. Transformations may have to be performed manually.

The following points must be taken into account during a later transformation:

- Replacement of standard role classes (e.g. RoleClass Port), which are no longer included in version 2.1 by equivalent model elements (e.g. InterfaceClass Port).
- Definition and replacement of attribute semantics by attribute types from the standard attribute type library of version 2.1.
- Enhancement of class instances based on enhancements of new versions of the instantiated classes.
- Transformation of the examples.

In the opinion of the authors, an upward transformation is easier to perform and more tolerable than the creation of libraries on the basis of library versions that have not yet been released, have not been agreed, and are partly missing.

Thus we only apply AutomationML 2.0. in this version of the document.

3 Representation in AutomationML

3.1 Structuring of AutomationML Components and Composite Components

AutomationML Components and Composite Components are modelled using the file format AutomationML and its object-oriented modelling mechanisms. All information to an AutomationML Component or Composite Component is stored as AutomationML object and may comprise a hierarchical sub structure. The modeling architecture for AutomationML Components or Composite Components is a distributed file architecture by using standard AutomationML capabilities to store data of multiple known or unknown domains by referencing known or unknown external files. The AutomationML model of an automation component or automation system is called AutomationML Component.

Regarding the structuring of AutomationML Component or AutomationML Composite Component information in AutomationML, the following provisions apply:

- Each AutomationML Component or Composite Component shall have a dedicated root element in the AutomationML file. For modelling an automation component type, the root element shall be a SystemUnitClass, and for individual automation components, the root object shall be an InternalElement. The root element shall be a SystemUnitClass or InternalElement and reference the RoleClass "AutomationComponent" defined in the standard role class lib "AutomationMLComponentBaseRCL".

- If an AutomationML Component is part of an AutomationML Composite Component the root of this component shall be an InternalElement.

Note 1: This InternalElement should be an instance of a SystemUnitClass directly or indirectly derived by an "AutomationComponent".

- Automation component information that belongs to one automation component shall be stored directly at the root element of the AutomationML Component or at one child element of it.

Note 2: All child elements belong to the root element and can be nested without any limit in depth and structuring.

- Child elements describing an information aspect of the AutomationML Component or Composite Component shall have a SupportedRoleClass defined in the standard role class lib "AutomationMLComponentBaseRCL" or "AutomationMLComponentStandardRCL" or any role class derived from a role class defined within this library.

Note 3: It is allowed to store any other additional AutomationML elements as child elements of the root element or one of its child elements. These AutomationML elements do not belong to the AutomationML Component definition within the AutomationML Component meta model.

Note 4: Between an AutomationML Component or Composite Component object and its child element there are no other InternalElements also referencing the role class AutomationComponent as RoleRequirements or SupportedRoleClass.

Figure 10 gives an overview of an AutomationML Component that contains its root element and four child elements that contain information to the described automation component.

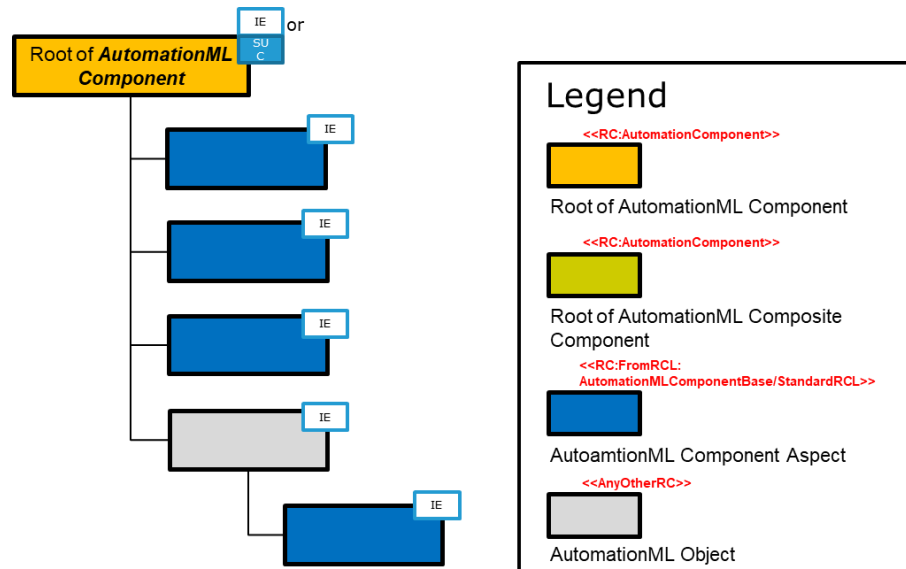


Figure 10: Overview structure of an automation component in AutomationML

This document does not define a certain hierarchical structure of the AutomationML Component model. An electric interface or an Icon may be modelled as a direct child of the AutomationML Component or may be positioned as a child in a deeper hierarchy. The identification of the objects happens by the associated role, whereas the hierarchical position of a model aspect has no further semantic.

3.2 Composition of AutomationML Automation Components

3.2.1 General Provisions

An automation system or component may consist of a composition of automation components. For the composition of such systems or components no structuring limitations exist made for the representation of AutomationML Composite Components.

Regarding the assignment of information to automation systems or components following provisions apply:

- Information aspect of an automation system or component shall be modelled as InternalElement that has a role requirement of a role class defined in the standard role class libs "AutomationMLComponentBaseRCL" or "AutomationMLComponentStandardRCL" or derived from these role classes.
- Each InternalElement within automation system or component information aspect shall be assigned to next higher InternalElement with the role requirement "AutomationComponent" or SystemUnitClass with the supported role class "AutomationComponent".

3.2.2 Example simple AutomationML Component

Figure 11 depicts an example of an AutomationML Component. The example contains an AutomationML Component "Component 1" that consist of the root element and three child InternalElements that contain information aspects. The root element may be an InternalElement with the role requirement "AutomationComponent" for instance models or SystemUnitClass with the supported role class "AutomationComponent" for type models.

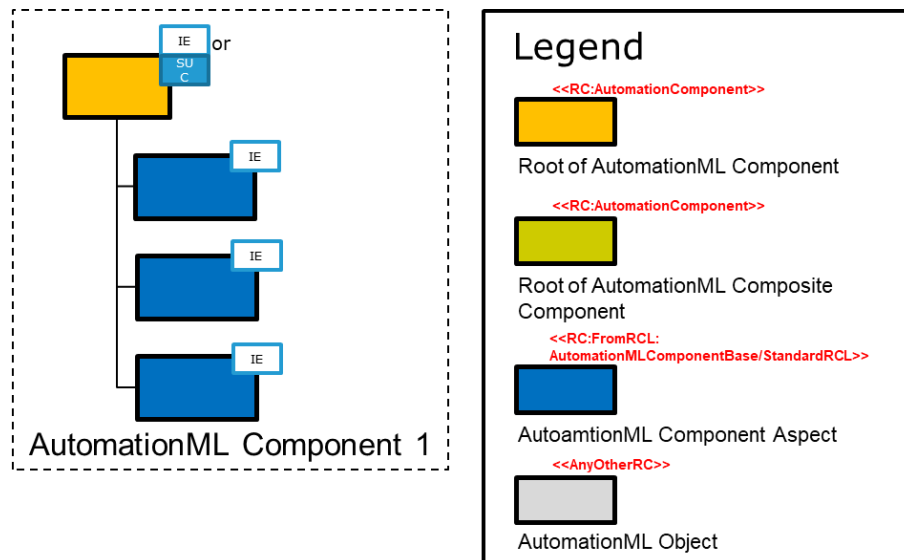


Figure 11: Example simple AutomationML Component

3.2.3 Example nested AutomationML Component with AutomationML Objects

Figure 12 shows an example that contains the AutomationML Component “Component 2”. This AutomationML Component consist of the root element, three child InternalElements that contain information aspects and an additional AutomationML object. Within the example the InternalElements and the AutomationML object are nested.

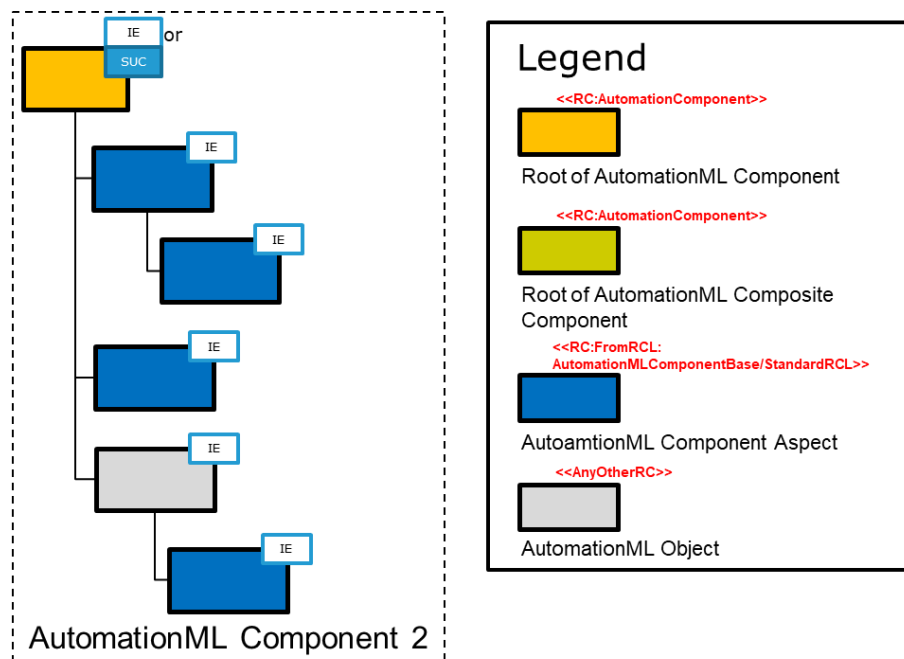


Figure 12: Example nested AutomationML Component with AutomationML Objects

3.2.4 Example AutomationML Composite Component

Figure 13 shows an example of an AutomationML Composite Component. The AutomationML Composite Component has one AutomationML Component as sub-component. Additional tree information aspects are attached to the overall AutomationML Composite Component and one information aspect is attached to the sub AutomationML Component.

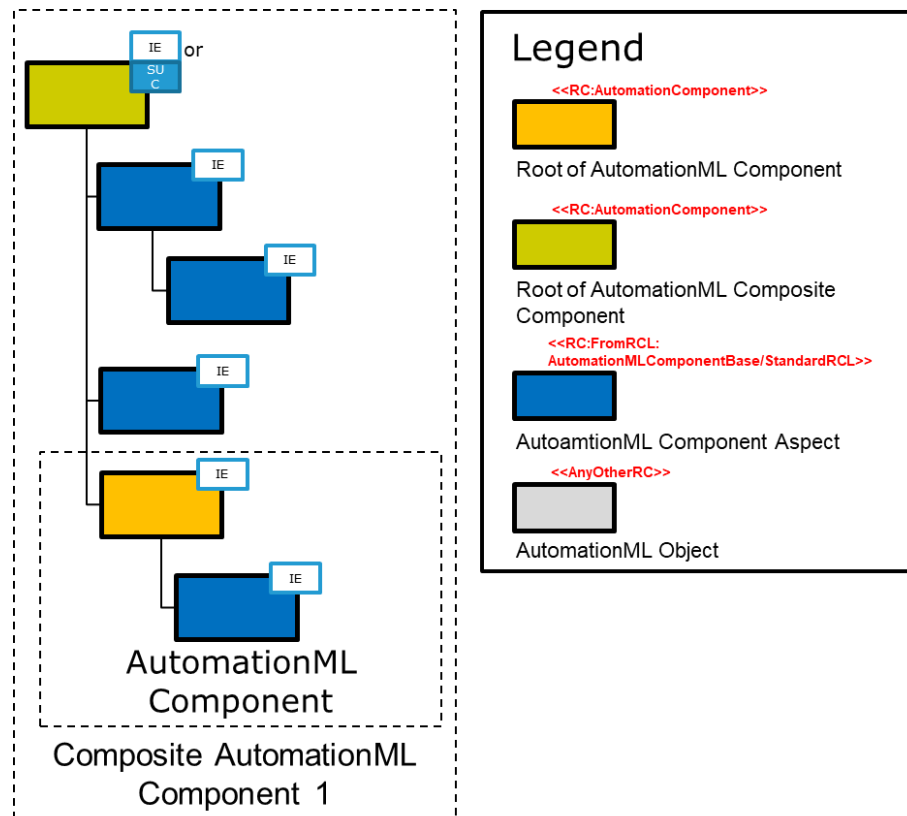


Figure 13: Example AutomationML Composite Component

3.2.5 Example nested AutomationML Composite Component with hierarchies

Figure 14 shows an example of an AutomationML Composite Component that has two sub AutomationML Components. Within this example the AutomationML Composite Component has integrated hierarchies and is nested.

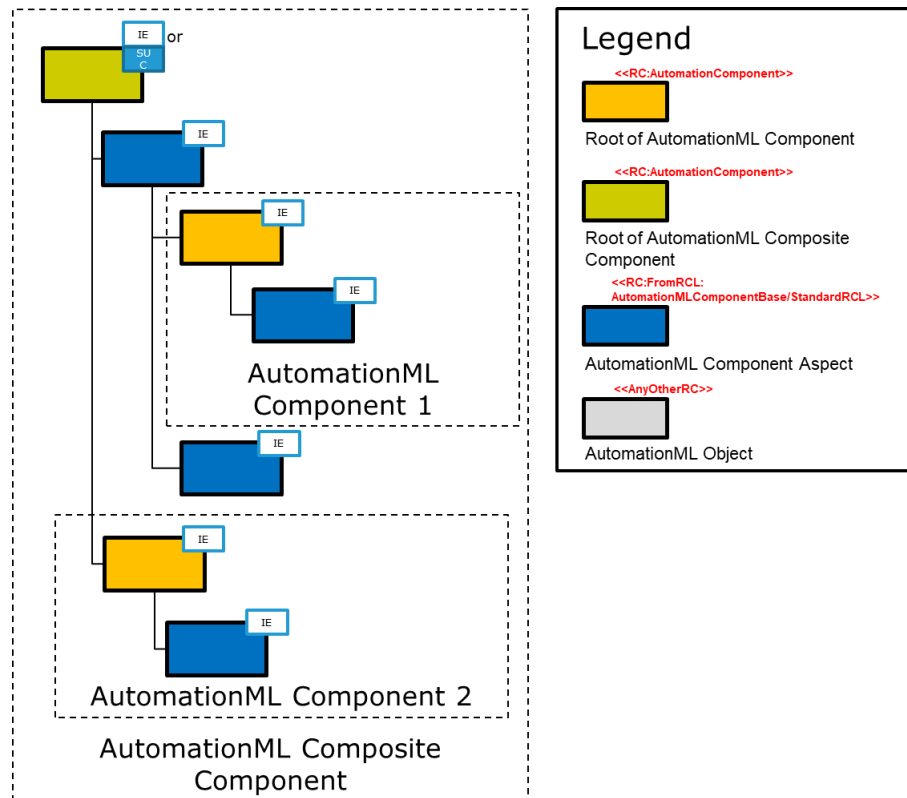


Figure 14: Example nested AutomationML Composite Component with hierarchies

3.3 Mapping of General Data

General data of AutomationML Components or Composite Components are mapped to attributes of the root element of the AutomationML Component or Composite Component. To structure the attributes following main categories attributes are defined.

- Identification data
This section contains all attributes to unambiguously identify an automation component (including type and instance).
- General technical data
This section contains all classifiable data of the component.
- Commercial data
This section contains order and purchasing information.
- Parameter Data
This section should contain configuration information for instances (e.g. IP Address).

Note: The attribute definition is align with the “VendorNameplate” concept of OPC UA Device Integration companion specification [OPCUA-Part100:2020]

The categories of the attributes are defined as attributes of the role class “AutomationComponent”, see Figure 15: Representation of attribute groups as AutomationML attributes as UML diagram Figure 15 and Figure 16 shows the UML representation of this role class and within the AutomationML.

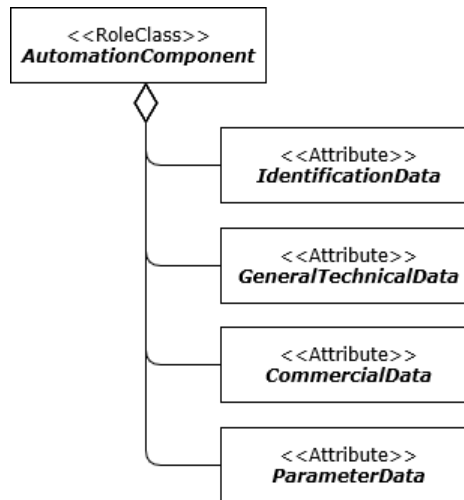


Figure 15: Representation of attribute groups as AutomationML attributes as UML diagram

AutomationMLComponentStandardRCL

RC AutomationComponent {**Class:** AutomationMLBaseRole }

Name	DataType
IdentificationData	Empty
GeneralTechnicalData	Empty
CommercialData	Empty
ParameterData	xs:string

Figure 16: AutomationComponent role class with attribute group definition

For the mapping of general data following rules shall apply:

- General data of an AutomationML Component or Composite Component shall be defined as attributes of the root element.

Note 1: If possible, the attributes should be stored as child elements of the attribute group defined in this chapter.

- The attribute shall be used according the definition of chapter 0
- The unit definition of attributes shall follow the provisions of the [BPR-Units:2018].

An extended example for the definition of general data according to IEC 62683-1 can be found in chapter 9.8.

3.4 General Model Integration

An AutomationML InternalElement representing an automation component as AutomationML Component or Composite Component may contain or reference model information to be used in different application scenarios like requirement specification, virtual commissioning or maintenance. Therefore, the representation of an automation component shall provide the ability to attach different partial models.

The model information should be described in a declarative manner without strong relations to a specific usage scenario. This approach enables the generic usage of the model information for different applications. This means for example, that in context of the application scenario “virtual commissioning” different models are used like the simulation model of the automation component and its geometric and kinematic models. But the same models may also be used for different application scenarios like floor planning and plant construction.

The following rules shall apply:

- Partial model of an AutomationML Component or Composite Component representation of an automation component shall be attached to an InternalElement, which shall be a child or sub child element of the root element of AutomationML Component.
- The InternalElement shall reference the role class “Model” of the role class library “AutomationMLComponentBaseRCL” or a derived role class.

Figure 17 depicts the general inheritance structure of the used role classes to integrate partial models. Additionally, Figure 18 depicts base role class “Model” as object tree.

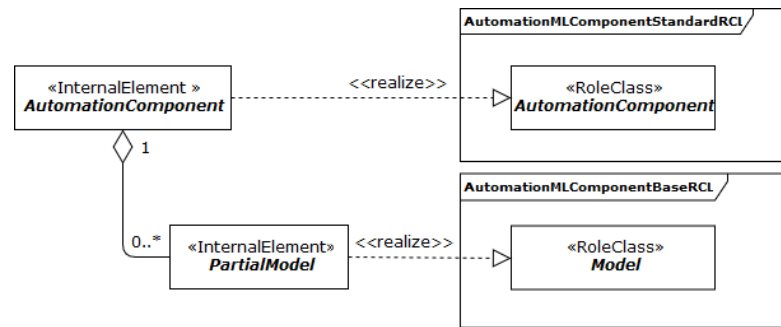


Figure 17: Example an AutomationML Component with one “Model”

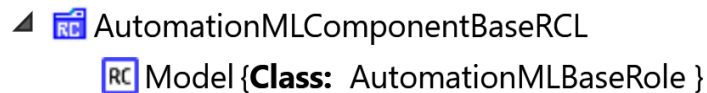


Figure 18: AutomationML Representation of “Model” role class

3.5 AutomationML Component Logic Representation

3.5.1 General

Automation components and systems are resources of an automated system that might be able to execute a certain task or process in a production. Often the automation components are controlled subsequently from one or many points for example by a PLC to achieve the desired behavior or process of production.

This chapter describes how to model the logical elements of an automation component as AutomationML Component that can be called from other logical entities of an automated system. Additionally, the integration of simulation models describing the behavior of the automation component based on logic description is part of this chapter.

In general, three different standardized logic models are supported for the description of a logic representation and simulation of automation component. They are differing in integration and usage within the AutomationML Component and the libraries they are based on. Figure 19 shows these models.

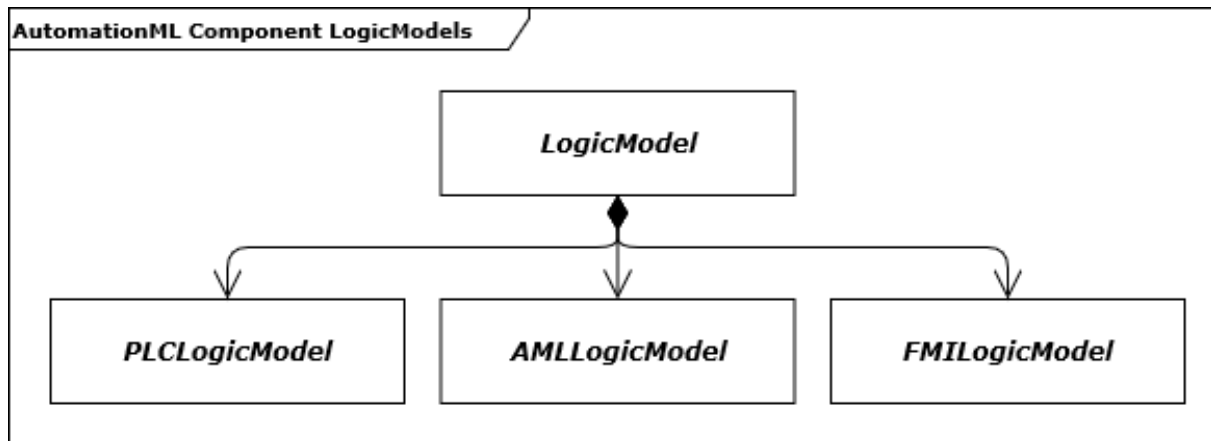


Figure 19: AutomationML Component logic models

The three models are:

- PLCLogicModel that defines logic description on base of [IEC61131-10:2019] and the integration of these models into AutomationML as defined in AutomationML Part4
- AMLLogicModel that defines logic description and integration into AutomationML on base of AutomationML [WP-Part4:2018].
- FMILogicModel that defines logic description on base of the FMI standard [FMI:2019] the integration of this model into AutomationML is defined within this Whitepaper.

These three possible models shall support different use cases for logics of an AutomationML Components that might be realized by any of them and not be restricted to a certain kind of logic representation. These uses cases for the model integration are in general the description of

- Behaviour models
- Simulation models
- Functions
- Sequences or sequence elements,
- and Skill logic models.

3.5.2 Logic Models

To integrate the tree logic models PLCLogicModel, AMLLogicModel and FMILogicModel in the context of an AutomationML Component description four role classes are defined in the role class library "AutomationMLComponentBaseRCL". The role classes are:

- "LogicModel" as abstract role class for logic description. This role class can be used for the integration of own unspecific logic models.

Note 3: The specific logic models are not derived from "LogicModel" because they bring their own parent role class defined in [WP-Part1:2018], [WP-Part4:2018] and the role class library AutomationMLFMILogicRoleClassLib" see 4.1.4.

- "PLCopenXMLLogic" as specific role class for the integration of PLCopen XML based PLC logic models.

Note 1: The general integration of PLCopen XML models in AutomationML is defined in [WP-Part1:2018] and [WP-Part4:2018].

- "AMLLogic" specific role class for the integration of AMLLogic based logic models.

Note 2: The general integration of AMLLogic models in AutomationML is defined in [WP-Part4:2018].

- "FMILogic" specific role class for the integration of FMI based logic models according to [FMI:2019].

Figure 20 shows the role classes for integration of logic models as AutomationML tree. Additional Figure 21 depicts the used interface classes within the role class definition. The inheritance structure between the role classes for the logic description and basic AutomationML role classes as well as the relation to the different interface classes is shown in Figure 22.



- ▲ **RC** LogicModel {**Class:** Model }
 - ▲ **RC** PLCOpenXMLLogic {**Class:** LogicModelObject }
 - ▲  PLCOpenXMLLogic-Interfaces
 - VariableInterface {**Class:** VariableInterface }
 - LogicInterface {**Class:** LogicInterface }
 - RC** AMLLogic {**Class:** LogicModelObject }
 - ▲ **RC** FMILogic {**Class:** FMILogicObject }
 - ▲  FMILogic-Interfaces
 - FMIReference {**Class:** FMIReference }
 - FMIVariableInterface {**Class:** FMIVariableInterface }

Figure 20: Role Classes for automation component Logic Model

- ▲ **ICL** AutomationMLInterfaceClassLib
 - ▲ **IC** AutomationMLBaseInterface
 - ▲ **IC** ExternalDataConnector {**Class:** AutomationMLBaseInterface }
 - ▲ **IC** PLCOpenXMLInterface {**Class:** ExternalDataConnector }
 - ▶ **IC** VariableInterface {**Class:** PLCOpenXMLInterface }
 - ▲ **ICL** AutomationMLPLCOpenXMLInterfaceClassLib
 - IC** VariableInterface
 - ▲ **ICL** AutomationMLFMILogicInterfaceClassLib
 - IC** FMIReference {**Class:** ExternalDataReference }
 - IC** FMIVariableInterface {**Class:** ExternalDataConnector }
 - ▲ **ICL** AutomationMLLogicInterfaceClassLib
 - IC** SequencingLogicModelInterface {**Class:** LogicModelInterface }
 - IC** BehaviourLogicModelInterface {**Class:** LogicModelInterface }
 - IC** VariableInterface {**Class:** LogicModelElementInterface }

Figure 21: Interface Classes for automation component Logic Model

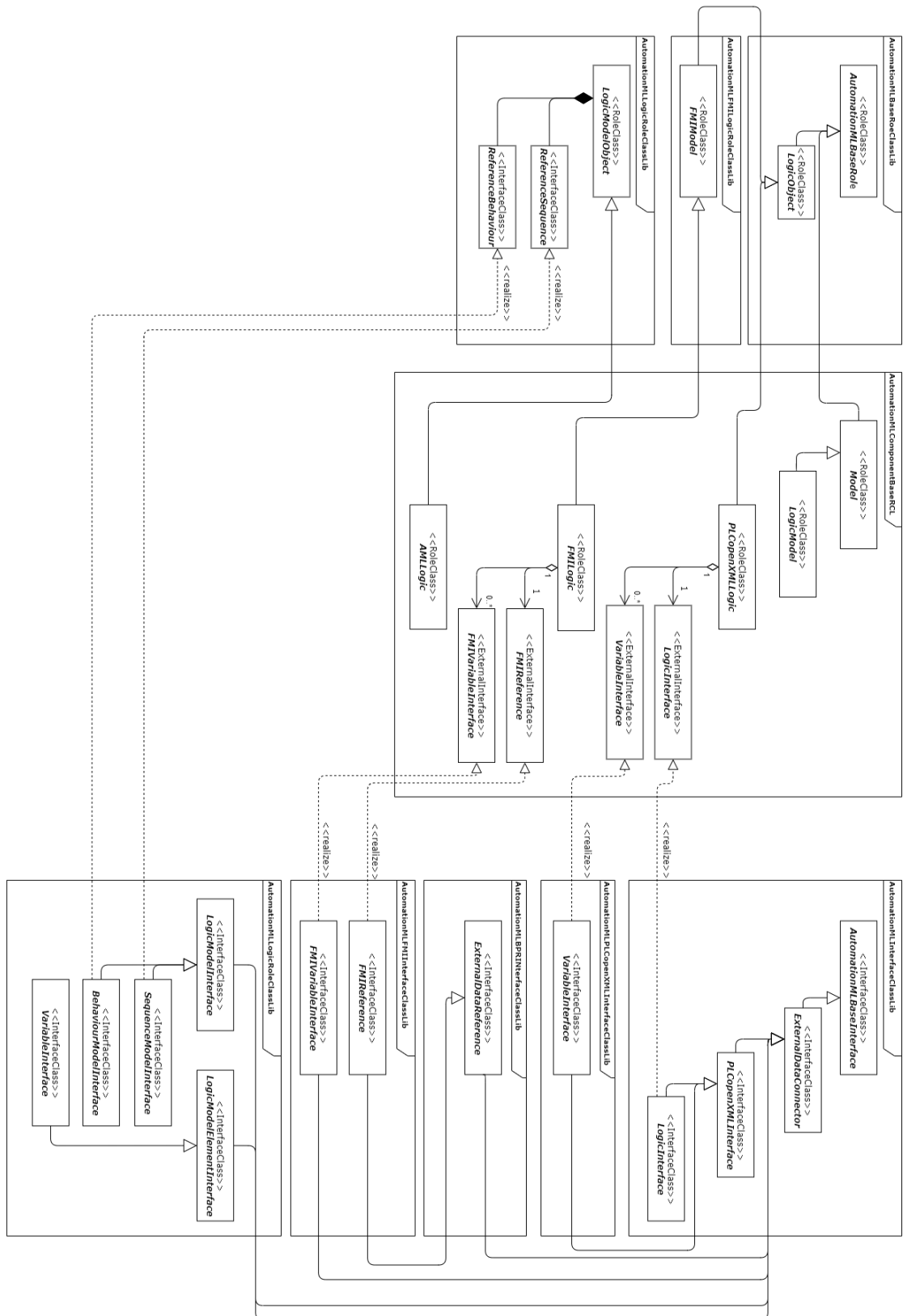


Figure 22: Inheritance structure for an AutomationML logic model integration

3.5.3 Usage of Logic Models

For the usage of in logic models in a specific context additional to role classes that define the model type, roles that specify the context are needed. These role classes are “BehaviourModel”, “SimulationModel”, “Function”, “Sequence”, “SequenceElement” and “SkillLogicModel”. They are defined in the role class library “AutomationMLComponentStandardRCL”. For these role classes and their use following provisions apply:

- The role classes shall be a child of the role class “LogicModel” of the role class library “AutomationMLComponentBaseRCL” or derived role class.
- An InternalElement that has a reference to one of the role classes shall have a second reference to a role class defined in chapter 3.5.2 or a child of these role classes.

Figure 23 shows role classes that shall be used to define the context of a logic model integration as AutomationML tree additional the Figure 24 depicts the inheritance structure of these role classes.

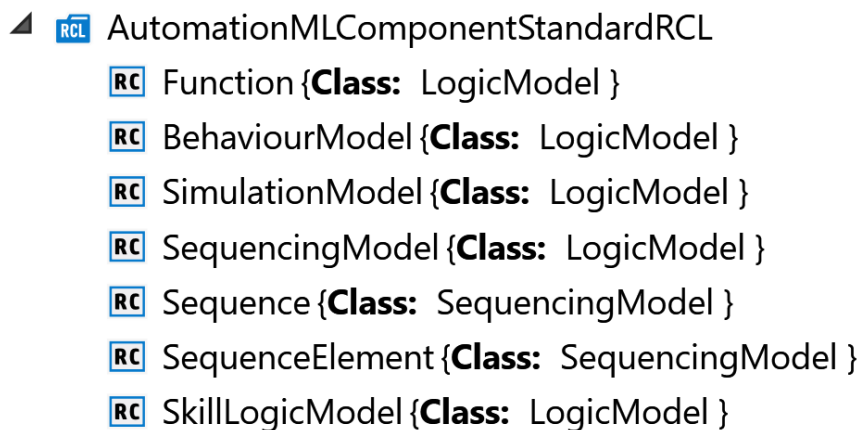


Figure 23: Role Classes for automation component logic use cases

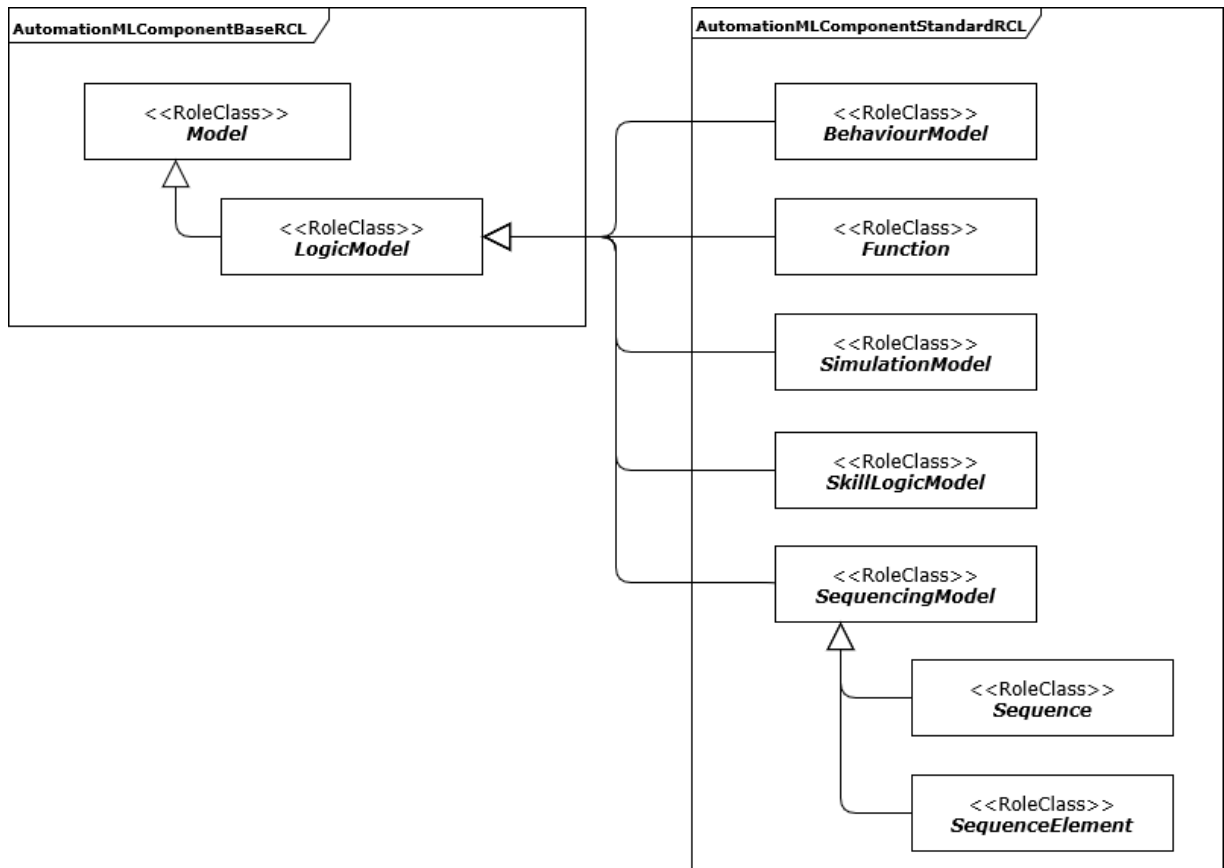


Figure 24: Inheritance structure for an AutomationML logic use cases

Figure 25 shows an example. Within this Example an InternalElement “FMILogicModel” realize the role class “SimulationModel” in combination with the role class “FMILogic”. This combination of role classes has following result. The InternalElement shall be used to integrate a simulation model and the integrated model shall be an FMI model.

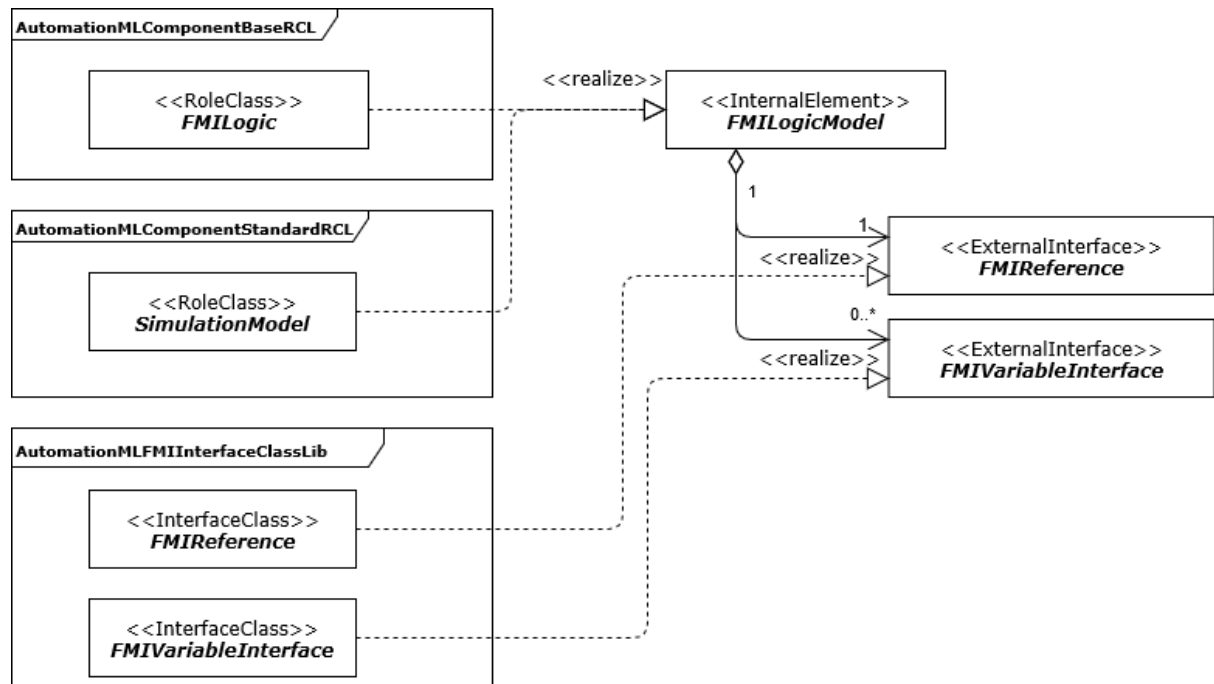


Figure 25: Example integration of an FMI Logic Model

3.5.4 Behaviour

One model aspect of automation components and composite components is related to their behavior. An InternalElement with the role class “BehaviourModel” or a derived role class shall be used in order to refer the behaviour model.

Following provisions shall be applied in order to integrate the behaviour model of the automation component into an AutomationML file:

- Behaviour model integration information of an AutomationML Component shall be attached to an InternalElement.
- The InternalElement carrying the behavior model integration information shall reference the role class “BehaviourModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the behavior model integration information shall reference the role class “LogicModel”, “PLCopenXMLLogic”, “AMLLogic” or “FMILogic” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the behavior model shall be a child or sub child element of that AutomationML element representing the automation component.

Note 1: Each AutomationML Component representing an automation component may have references to zero or more behaviour models.

Note 2: An automation component can be represented by different behaviour models. Where each behaviour model is represented by its own file. These different behaviour models can be related to different environments. The InternalElement carrying the behavior model information then will differ in the referenced role classes, which must be derived role classes from the role class “BehaviourModel”.

- If the complete behaviour model is stored in separated files, then each file shall have a separate InternalElement carrying the behaviour model integration.

Note 3: If the files are linked within their specification scope one reference to the root of the behaviour model shall be enough.

Figure 26 depicts the general inheritance structure of the role classes “BehaviourModel”. Additional Figure 27 shows the role class as object tree.

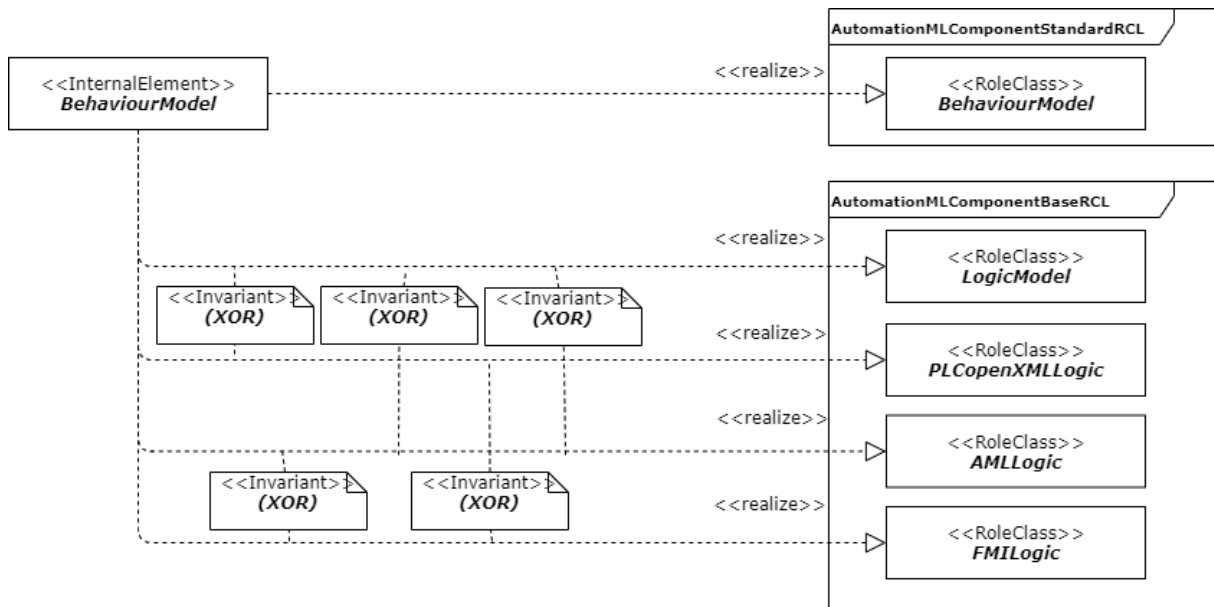


Figure 26: Inheritance structure for an AutomationML “BehaviourModel”

AutomationMLComponentBaseRCL

BehaviourModel {Class: Model }

Figure 27: AutomationML Representation of “BehaviourModel” role class

3.5.4.1 PLCopen XML based Behaviour Model

For the integration of PLCopen XML based behaviour models following provisions apply:

- A PLCopen XML based behaviour model of an automation component shall be referenced by an InternalElement carrying the behaviour information.
- This InternalElement shall realize the role class “PLCopenXMLLogic” of the role class library “AutomationMLComponentBaseRCL”
- This InternalElement shall realize the role class “BehaviourModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- This InternalElement shall be a child or sub child element of the AutomationML element representing the automation component.
- An ExternalInterface with the interface class “LogicInterface” of the interface class lib “AutomationMLInterfaceClassLib” shall be used to refer the model.
- All provisions from Part 4 to reference logic shall apply.
- Each published variable of the PLCopen XML based behaviour model shall be referenced with the interface type “VariableInterface” of the interface type library “AutomationMLPLCopenXMLInterfaceClassLib”.

Figure 28 shows the example usage of the role classes “PLCopenXMLLogic” and “BehaviourModel” to reference a PLCopen XML based logic model as behavior description to an internal element. Figure 29 shows additionally an example for the integration of a PLCopen XML behavior model as AutomationML tree.

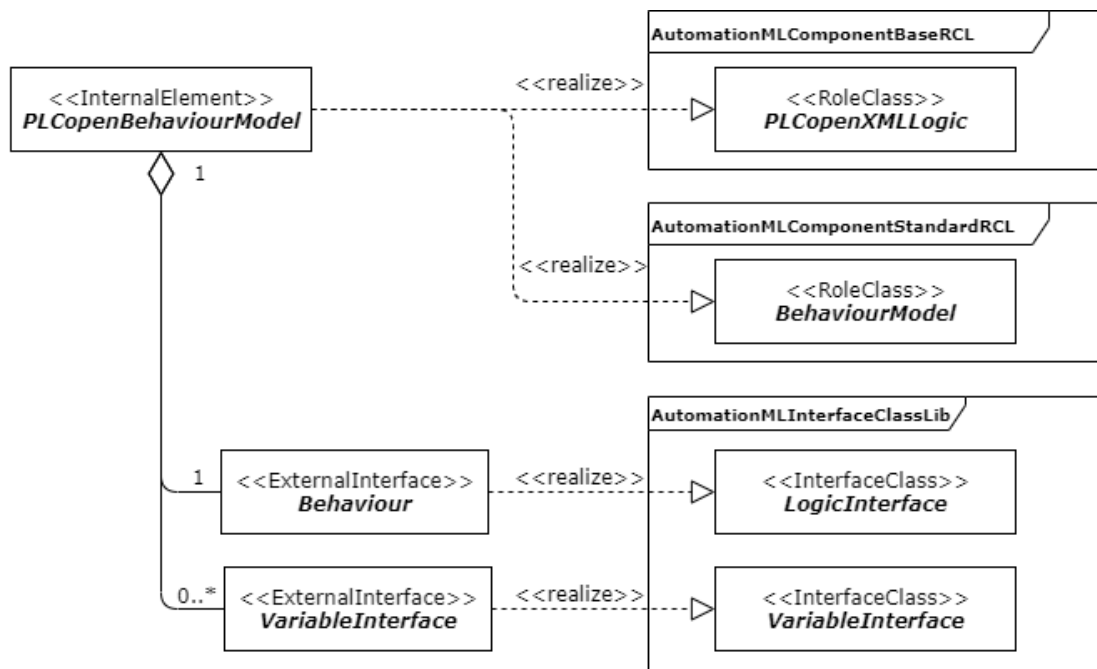


Figure 28: Example usage of the AutomationML role class “PLCLogic” and “BehaviourModel” as AutomationML “PLCopenBehaviourModel”

- ▲ **IE** BehaviourModel in PLCopenXML {**Role:** BehaviourModel, PLCopenXMLLogic}
 - ▲ **RR** BehaviourModel in PLCopenXML-Interfaces
 - VariableInterface {**Class:** VariableInterface }
 - LogicInterface {**Class:** LogicInterface }
 - SRC** AutomationMLComponentBaseRCL/LogicModel/PLCopenXMLLogic
 - RR** AutomationMLComponentStandardRCL/BehaviourModel

Figure 29: Example AutomationML Representation of “PLCopenBehaviourModel”

3.5.4.2 AMLLogic based Behaviour Model

For the integration of AMLLogic based behaviour models following provisions apply:

- An AMLLogic based behaviour model of an automation component shall be referenced by an InternalElement carrying the behaviour information.
- This InternalElement shall realize the role class “AMLLogic” of the role class library “AutomationMLComponentBaseRCL”
- This InternalElement shall realize the role class “BehaviourModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- This InternalElement shall be a child or sub child element of the AutomationML element representing the automation component.
- An ExternalInterface with the interface class “BehaviourLogicInterface” of the interface class lib “AutomationMLLogicInterfaceClassLib” shall be used to refer the model.
- All provisions from Part 4 to reference logic shall apply.
- Each published variable of the AMLLogic based behaviour model shall be referenced with the interface type “VariableInterface” of the interface type library “AutomationMLComponentBaseCL”.

Figure 30 shows the example usage of the role classes “AMLLogic” and “BehaviourModel” to reference AMLLogic based logic model as behavior description to an internal element. Figure 31 shows additionally an example for the integration of an AMLLogic model as AutomationML tree.

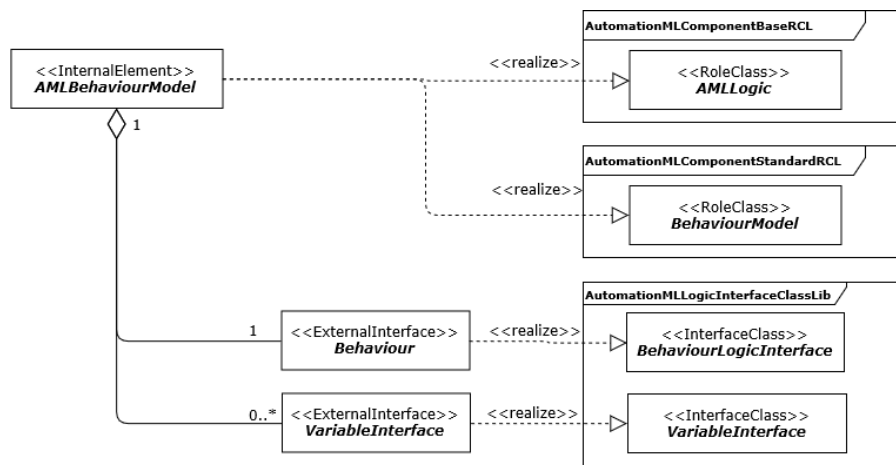


Figure 30: Example usage of the AutomationML role class “AMLLogic” and “BehaviourModel” as AutomationML “AMLBehaviourModel”

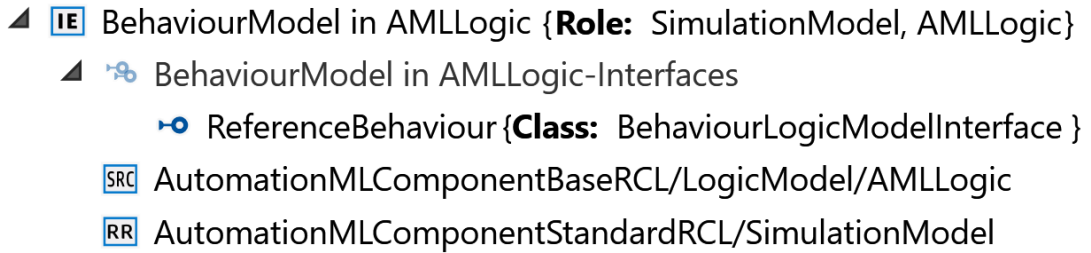


Figure 31: Example AutomationML Representation of “AMLBehaviourModel”

3.5.4.3 FMI based Behaviour Model

Following provisions shall be applied for the integration of FMILogic models:

- An FMILogic model of an automation component shall be referenced by an InternalElement carrying the behaviour integration information.
- This InternalElement shall realize the role class “FMILogic” of the role class library “AutomationMLComponentBaseRCL”.
- This InternalElement shall realize the role class “BehaviourModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- This InternalElement shall be a child or sub child element of the AutomationML element representing the automation component.
- An ExternalInterface with the interface class “FMIReference” shall be used to refer the FMI-based simulation model, which is called a Functional Mockup Unit (FMU).
- All published variables of the FMILogic shall be referenced with the interface class “FMIVariableInterface” of the interface type library “AutomationMLFMIInterfaceClassLib”.

Figure 32 shows the example usage of the role classes “FMILogic” and “BehaviourModel” to reference FMI based logic model as behavior description to an internal element. Figure 33 shows additionally an example for the integration of an FMI behavior model as AutomationML tree.

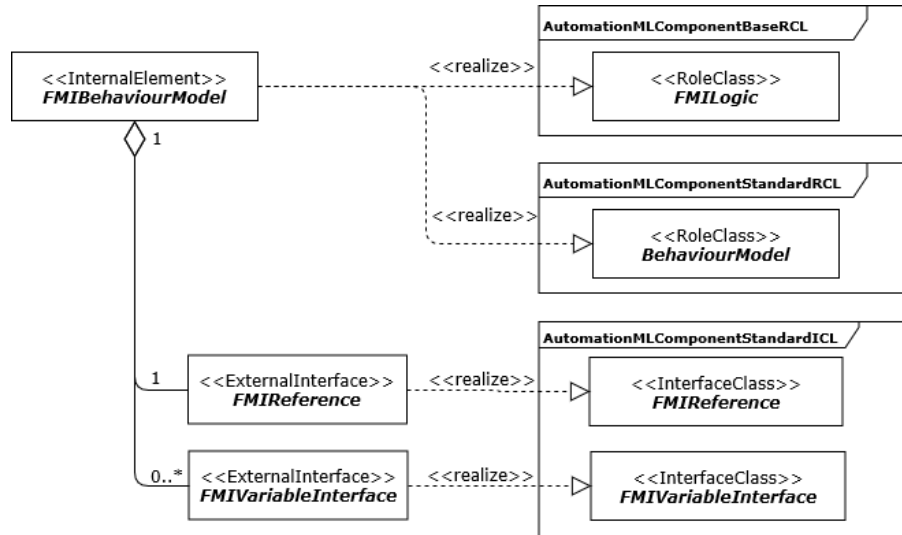


Figure 32: Example usage of the AutomationML role class “FMILogic” and “BehaviourModel” as AutomationML “FMIBehaviourModel”

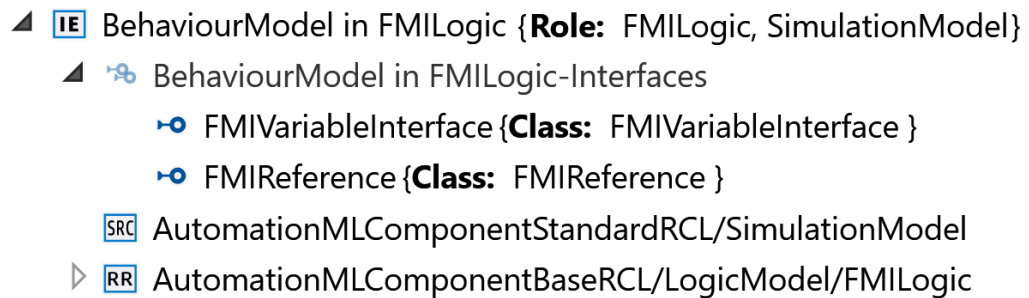


Figure 33: Example AutomationML Representation of “FMIBehaviourModel”

3.5.5 Simulation

One model aspect of automation components and systems is related to their simulation models. An InternalElement with the role class “SimulationModel” or a derived role class shall be used in order to refer the simulation model.

Following provisions shall be applied in order to integrate the simulation model of the automation component into an AutomationML file:

- Simulation model information of an AutomationML Component shall be attached to an InternalElement.
- The InternalElement carrying the simulation model integration information shall reference the role class “SimulationModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the simulation model integration information shall reference the role class “LogicModel”, “PLCopenXMLLogic”, “AMLLLogic” or “FMILogic” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the simulation model shall be a child or sub child element of that AutomationML element representing the automation component.

Note 1: Each automation component may have references to zero or more simulation models.

Note 2: An automation component can be represented by different simulation models each referring to its own file. These different behaviour models can be related to different environments. The InternalElement carrying the behavior model information then will differ in the referenced role classes, which must be derived role classes from the role class “SimulationModel”.

- If the complete simulation model is stored in separated files, then each file shall have a separate InternalElement carrying the simulation model integration.

Note 3: If the files are linked within their specification scope one reference to the root of the simulation model shall be enough.

Figure 34 depicts the general inheritance structure of the role classes “SimulationModel”. Additional Figure 35 shows the role class as object tree.

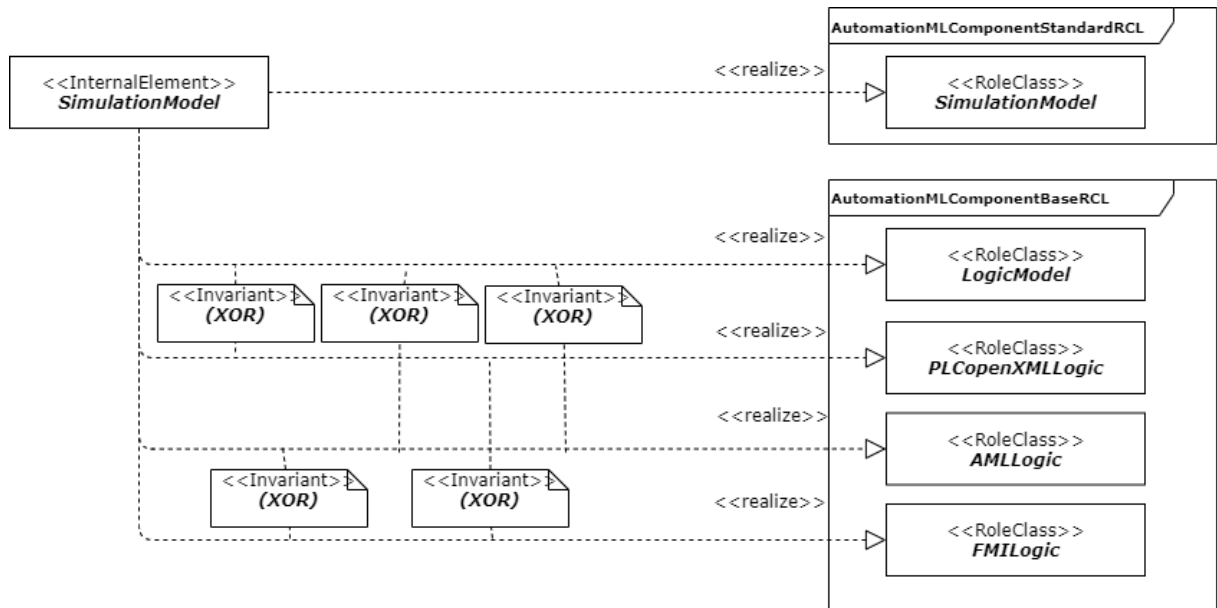


Figure 34: Inheritance structure for an AutomationML “SimulationModel”

AutomationMLComponentStandardRCL SimulationModel {**Class:** LogicModel }

Figure 35: AutomationML Representation of “SimulationModel” role class

3.5.5.1 PLCopen XML based Simulation Model

The same mapping rules, to integrate a PLCopen XML based “SimulationModel” as described for PLCopen XML based behaviour models as described in 3.5.4.1, are applied here too.

3.5.5.2 AMLLogic based Simulation Model

The same mapping rules, to integrate a AMLLogic based “SimulationModel” as described for AMLLogic based behaviour models as described in 3.5.4.2, are applied here too.

3.5.5.3 FMI based Simulation Model

The same mapping rules, to integrate a FMI based “SimulationModel” as described for FMI based behaviour models as described in 3.5.4.3, are applied here too.

3.5.6 Sequencing

Sequencing is used for any logical element that describes the instructions to the controlled automation component. Sequencing is class-divided in the role classes “Sequence” and “SequenceElement”. The role class “Sequence” is used for sequences of logic models that consists of one or many “SequenceElement”s.

The role class “SequenceElement” is used for objects that describe self-contained Functions that can be called in a sequence from a “Sequence” element.

Following provisions shall be applied in order to integrate the Sequencing model of the automation component into an AutomationML file:

- Sequencing model integration information of an AutomationML Component shall be attached to an InternalElement.
- The InternalElement carrying the sequencing model integration information shall reference the role class “Sequence” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.

Note 1: Each automation component may have references to zero or more sequencing models.

Note 2: An automation component can be represented by different sequencing models each referring to its own file. These different sequencing models can be related to different environments. The InternalElement carrying the sequence model information then will differ in the referenced role classes, which must be derived role classes from the role class “Sequence”.

- The InternalElement with role class with reference “Sequence” shall have zero to n child InternalElements. These InternalElements shall reference the role class “SequenceElement” of the role class library “AutomationMLComponentStandardRCL” or a derived role class. These InternalElements carrying the logic models for the single elements.
- The InternalElement carrying the sequence element model information shall reference the role class “LogicModel”, “PLCopenXMLLogic”, “AMLLogic” or “FMILogic” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the sequence element model shall be a child or sub child element of that AutomationML element representing the Automation Component.

Note 3: An automation component can be represented by different sequencing element models each referring to its own file. These different sequencing element models can be related to different environments. The InternalElement carrying the sequencing element model information then will differ in the referenced role classes, which must be derived role classes from the role class “SequenceElement”.

- If the complete sequence model is stored in separated files, then each file shall have a separate InternalElement carrying the sequence model integration.

Note 4: If the files are linked within their specification scope one reference to the root of the sequence model shall be enough.

Figure 36 depicts the general inheritance structure of the role classes “Sequence” and its derived role classes. Additional Figure 37 shows the role class as object tree.

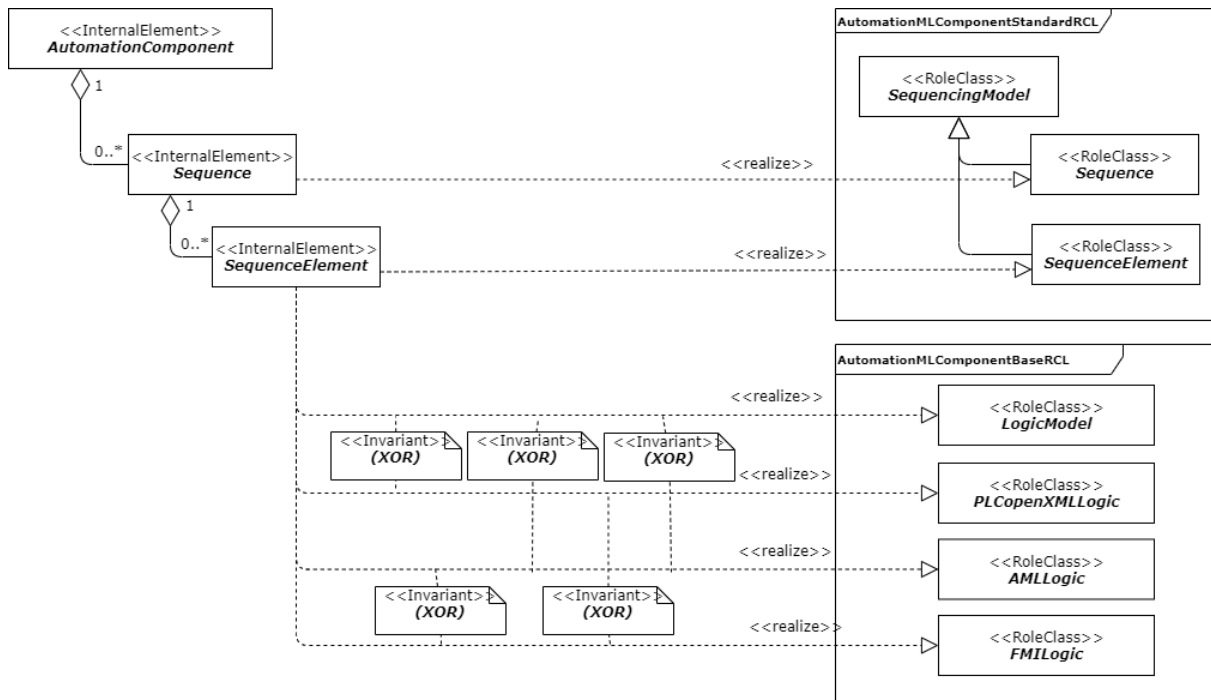


Figure 36: Inheritance structure of AutomationML “Sequence” role class

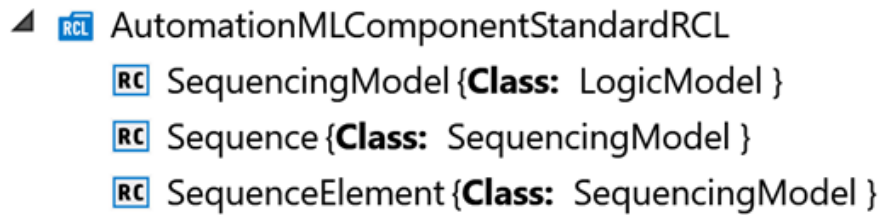


Figure 37: AutomationML representation of “Sequence” role class

3.5.6.1 PLCopen XML based SequencingElements

The same mapping rules, to integrate a PLCopen XML based “SequencingElement” as described for PLCopen XML based sequence models as described in 3.5.4.1, are applied here too.

3.5.6.2 AMLLogic based SequencingElements

The same mapping rules, to integrate a AMLLogic based “SequencingElement” as described for AMLLogic based sequence models described in 3.5.4.2, are applied here too.

3.5.6.3 FMI based SequencingElements

The same mapping rules, to integrate a FMI based “SequencingElement” as described for FMI based sequence models described in 3.5.4.3, are applied here too.

3.5.7 Function

One model aspect of automation components and systems is related to function that they can execute. An InternalElement with the role class “Function” or a derived role class shall be used in order to refer the function.

Following provisions shall be applied in order to integrate the function of the automation component into an AutomationML file:

- Function integration of an AutomationML Component shall be attached to an InternalElement.
- The InternalElement carrying the function integration information shall reference the role class “Function” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the function integration information shall reference the role class “LogicModel”, “PLCopenXMLLogic”, “AMLLogic” or “FMILogic” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the function shall be a child or sub child element of that AutomationML element representing the automation component.

Note 1: Each automation component may have references to zero or more functions.

Note 2: An automation component can be represented by different function models each referencing to its own file. These different functions can be related to different environments. The InternalElement carrying the function model information then will differ in the referenced role classes, which must be derived role classes from the role class “Function”.

- If the complete Function is stored in separated files, then each file shall have a separate InternalElement carrying the Function integration.

Note 3: If the files are linked within their specification scope one reference to the root of the function shall be enough.

Figure 38 depicts the general inheritance structure of the role classes “Function”. Additional Figure 39 shows the role class as object tree.

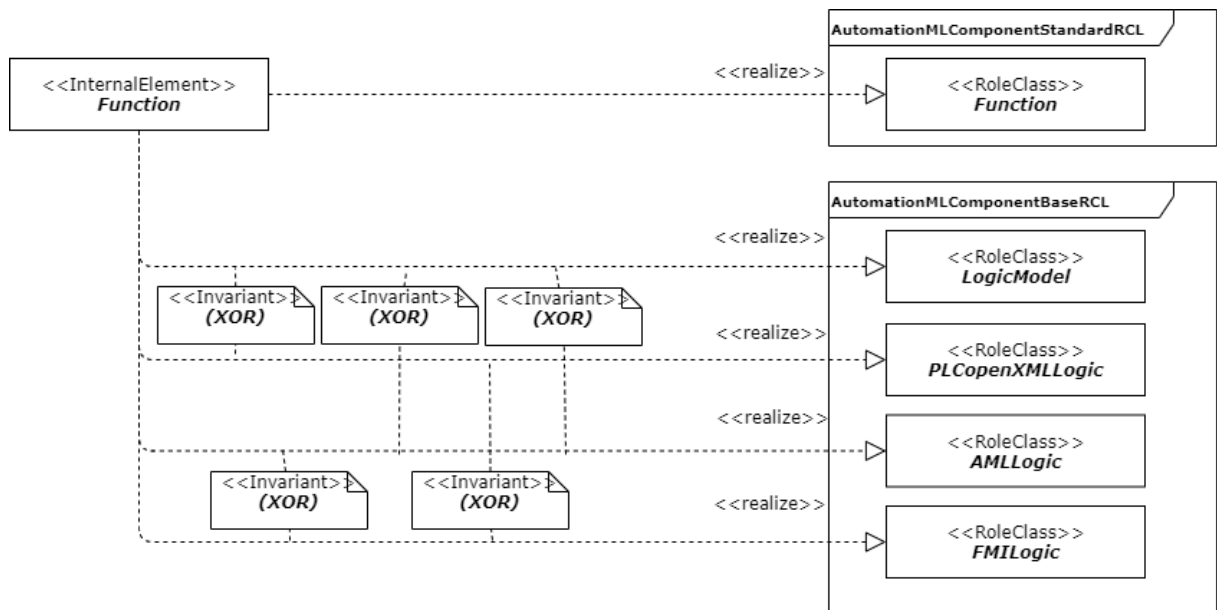


Figure 38: Inheritance structure for an AutomationML "Function"



Figure 39: AutomationML Representation of "Function" role class

3.5.7.1 PLCopen XML based Function

The same mapping rules, to integrate a PLCopen XML based functions as described for PLCopen XML based behavior models described in 3.5.4.1, are applied here too.

3.5.7.2 AMLLogic based Function

The same mapping rules, to integrate an AMLLogic based functions as described for PLCopen XML based behavior models described in 3.5.4.2, are applied here too.

3.5.7.3 FMI based Function

The same mapping rules, to integrate a FMI based functions as described for FMI based behavior models described in 3.5.4.3, are applied here too.

3.5.8 SkillsLogic

Skills are a standardized implementation of a self-contained Function of an automation component with the aim to be interoperable and interchangeable.

Skills can be composed to higher level skills and put together to a sequence in order to describe full processes in automation. The connection of skill logic models of different AutomationML Components is not part of this whitepaper. So, a basic definition for skill logic models will be presented.

Following provisions shall be applied in order to integrate the skill logic model of the automation component into an AutomationML file:

- Skill logic model information of an AutomationML Component shall be attached to an InternalElement.
- The InternalElement carrying the skill logic model shall reference the role class "SkillLogicModel" of the role class library "AutomationMLComponentStandardRCL" or a derived role class.

- The InternalElement carrying the behavior model integration information shall reference the role class “LogicModel”, “PLCopenXMLLogic”, “AMLLogic” or “FMLogic” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the skill logic model shall be a child or sub child element of that AutomationML element representing the automation component.

Note 1: Each automation component may have references to zero or more skill logic models.

Note 2: An automation component can be represented by different skill logic models each referring to its own file. These different skill logic models can be related to different environments. The InternalElement carrying the behavior model information then will differ in the referenced role classes, which must be derived role classes from the role class “SkillLogicModel”.

- If the complete skill logic model is stored in separated files, then each file shall have a separate InternalElement carrying the skill logic model integration.

Note 3: If the files are linked within their specification scope one reference to the root of the skill logic model shall be enough.

Figure 40 depicts the general inheritance structure of the used role class “Skill”. Additional Figure 41 shows the role class as object tree.

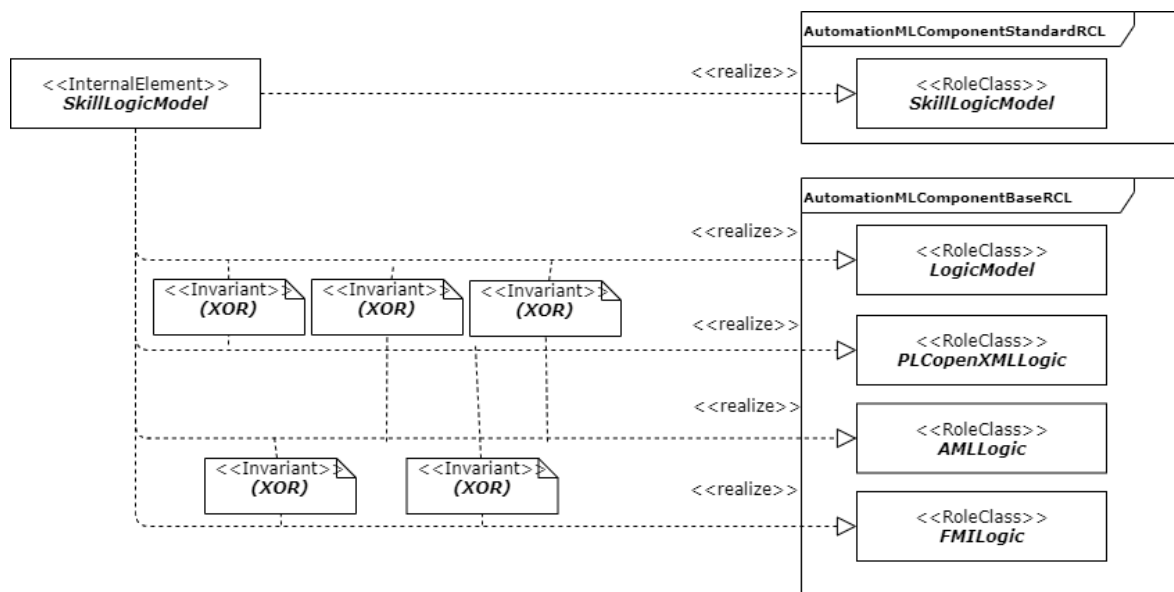


Figure 40: Inheritance structure for an AutomationML “SkillLogicModel”

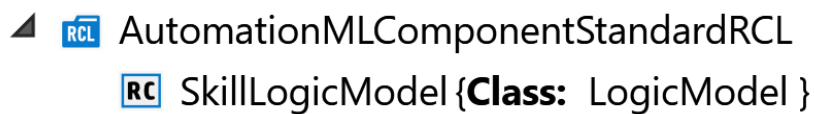


Figure 41: AutomationML Representation of “SkillLogicModel”

3.5.8.1 PLCopen XML based SkillLogicModel

The same mapping rules, to integrate a PLCopen XML based skill logic models as described for PLCopen XML based behavior models described in 3.5.4.1, are applied here too.

3.5.8.2 AMLLogic based SkillLogicModel

The same mapping rules, to integrate an AMLLogic based skill logic models as described for PLCopen XML based behavior models described in 3.5.4.2, are applied here too.

3.5.8.3 FMI based SkillLogicModel

The same mapping rules, to integrate an FMI based skill logic models as described for FMI based behavior models described in 3.5.4.3, are applied here too.

3.6 Geometry and Kinematic Model

3.6.1 Geometry

One model aspect of automation components and systems is the geometry model.

The following rules shall apply:

- The preferred format for modelling and exchanging geometry models shall be COLLADA
- Geometry model of an AutomationML Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component.
- The InternalElement shall reference the role class “GeometryModel” of the role class lib “AutomationMLComponentBaseRole” or a derived role class, which shall be a child or sub child element of the AutomationML element representing the automation component.

Note 1: Each automation component may have references to zero or more geometry models.

Note 2: The geometry model of a component may consist of 1 or n separate partial models that are stored in separate files.

- If the complete geometry model is stored in separated files, each file shall have a separate reference.

Note 3: If the files are linked within their specification scope one reference to the root of the geometry model shall be sufficient.

- Multiple geometry model formats can be used to represent the component, but they shall be integrally closed within each format and shall not reference each other.

Figure 42 depicts the general inheritance structure of the used role classes to integrate geometry models. Figure 43 shows the role class as object tree.

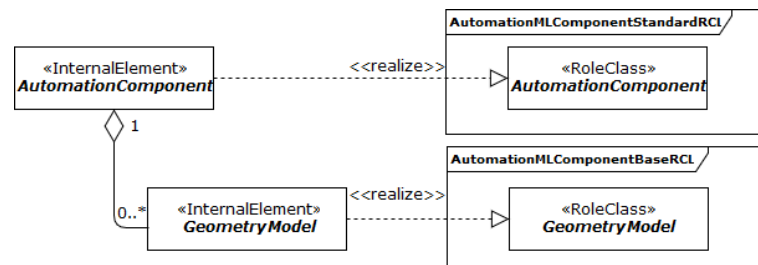


Figure 42: Example usage of the AutomationML role class “GeometryModel”

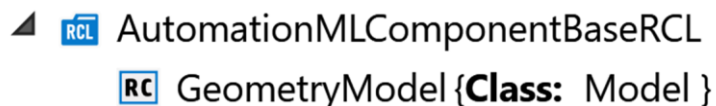


Figure 43: AutomationML representation of “Model” role class

3.6.1.1 Collada Geometry Model

For the integration of COLLADA the following rules shall apply:

- All specifications form [WP-Part3:2017] shall apply.
- A COLLADA geometry model of an automation component shall be attached with an “COLLADAInterface” to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component as AutomationML Component.

- The InternalElement shall reference the role class “COLLADAGeometryModel” of the role class lib “AutomationMLComponentBaseRCL” or a derived role class.
- If the InternalElement represents a coordinate system that shall be published, the Internal Element shall additionally reference the role class “Frame” from “AutomationMLBaseRoleClassLib”.
- If the InternalElement defines an offset in position to a direct parent COLLADAGeometryModel the attribute “Frame” with the appropriate values shall be added to that Internal Element

Note 1: If the attribute “Frame” is missing all sub attributes x, y, z, rx, ry, rz shall be assumed to be 0, see [WP-Part3:2017]

Note 2: The translation of the InternalElement by the attribute “Frame” relates to the parent InternalElement, SystemUnitClass, InstanceHierarchy or SystemUnitClassLibrary.

Note 3: The elements InstanceHierarchy and SystemUnitClassLib specify a three-dimensional, orthogonal, right-handed coordinate system with standard basis. The positive z axis is considered upward, the positive x direction defines the right axis and the negative y direction defines the forward axis.

- To refer the geometry model in a COLLADA document an ExternalInterface with the interface class “COLLADAInterface” shall be used that points to the COLLADA element visual_scene of an COLLADA document

Note 4: If one COLLADA document contains geometry, kinematics and the binding of both, the importer tool has to bind the COLLADAGeometryModel by matching over the same file name and the bind_kinematics_model of the COLLADA kinematics_scene

- The root COLLADA geometry model shall be an explicit COLLADA geometry model by setting the attribute refType of the ExternalInterface to “explicit”.

Note 5: An AutomationML Component might have zero or more root COLLADA geometry models

Note 6: A explicit COLLADA geometry model shall not be a child or sub child of an explicit COLLADA geometry model

Note 7: An explicit COLLADA geometry model usually references the whole geometry of an object

- The root of COLLADA geometry model might have zero or more “implicit” COLLADA geometry models or COLLADA geometry attachments as a child or a sub child.

Note 8: An implicit COLLADA geometry model usually references a detail of the explicit COLLADA geometry model

- A COLLADA geometry attachment is an InternalElement that shall reference the role class “COLLADAGeometryAttachment” that is derived from the role class “COLLADAGeometryModel” from role class library “AutomationMLComponentStandardRCL”.

Note 10: A COLLADA geometry attachment represents a connectable coordinate system

Note 11: A COLLADA geometry attachment refers to the next parent COLLADA geometry model

- The InternalElement of a COLLADA geometry attachment shall have one ExternalInterface with the InterfaceClass “COLLADAInterface” referencing the COLLADA visual_scene node and one ExternalInterface with the InterfaceClass “AttachmentInterface” of AutomationMLInterfaceClassLib.

Note 13: An AttachmentInterface can be connected with an InternalLink with other geometry objects in order to create a geometrical coupling.

Note 14: The direction of the InternalLink defines the direction of the coupling of the Internal Link's elements. The element on RefPartnerSideA moves the element on RefPartnerSideB, that means that a unidirectional connection with two InternalLinks is needed for a fixed geometrical coupling.

If the COLLADA geometry model belongs to a composition of automation components or an automation system and is not assigned to the root element of the automation component or system, the value “implicit” of the attribute “refType” of the interface shall be allowed.

Figure 44 depicts the general inheritance structure of the role class “COLLADAGeometryModel”. Additional Figure 45 shows the role class as object tree.

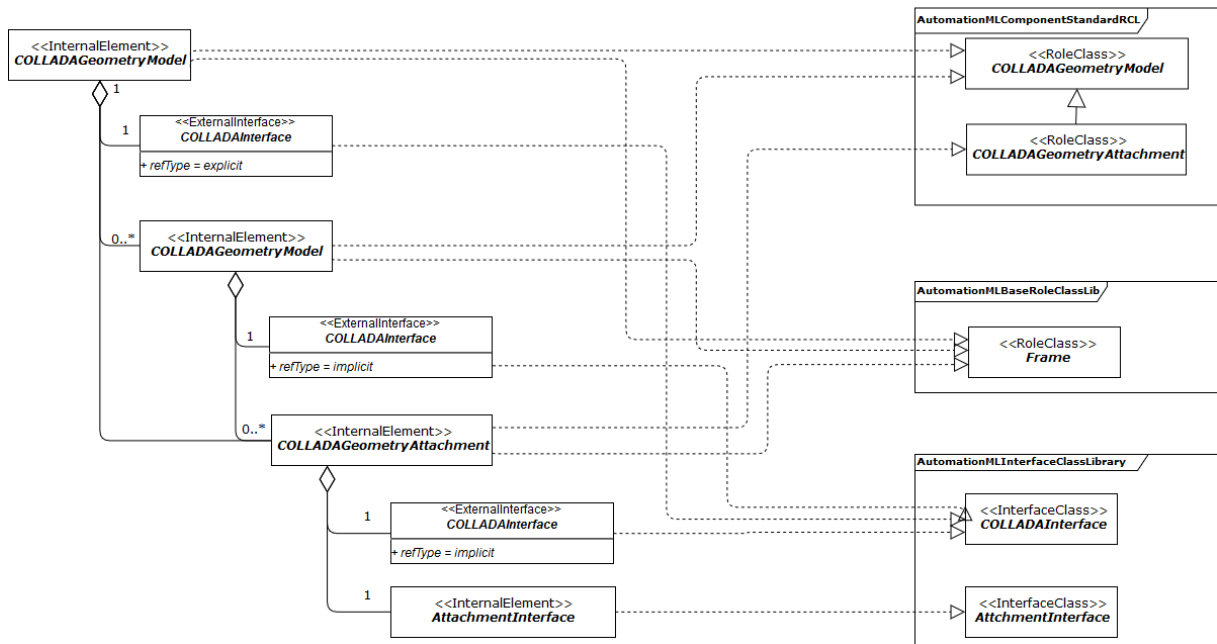


Figure 44: Example usage of AutomationML role classes to integrate an AutomationML "COLLADAGeometryModel"

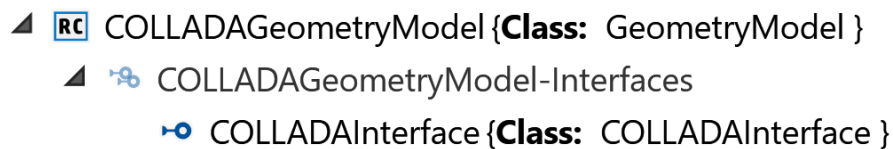


Figure 45: AutomationML Representation of "COLLADAGeometryModel" role class

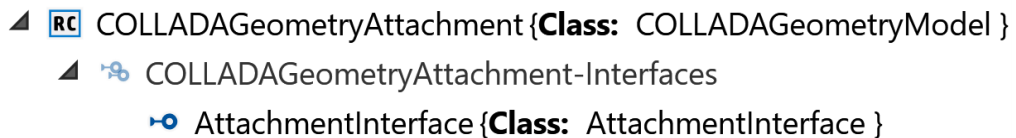


Figure 46: AutomationML Representation of "COLLADAGeometryAttachment" role class

3.6.1.2 JT Model

JT (Jupiter Tessellation) is an ISO-standardized 3D data format, used in industry for product visualization, collaboration and CAD data exchange, see [ISO 14306:2017]. For the integration of JT geometry models, the following rules shall apply:

- The reference of a JT Model for an automation component shall be attached as an InternalElement, which shall be a child or sub child element of the InternalElement representing the automation component as AutomationML Component.
- The InternalElement shall reference the role class "JTGeometryModel" of the role class lib "AutomationMLComponentBaseRCL" or a derived role class.
- To refer the model an ExternalInterface with the interface class "JTReference" of the interface class lib "AutomationMLComponentBaseICL" shall be used.

Figure 47 depicts the general inheritance structure of the role classes "JTGeometryModel". Additional Figure 48 show the role class as object tree.

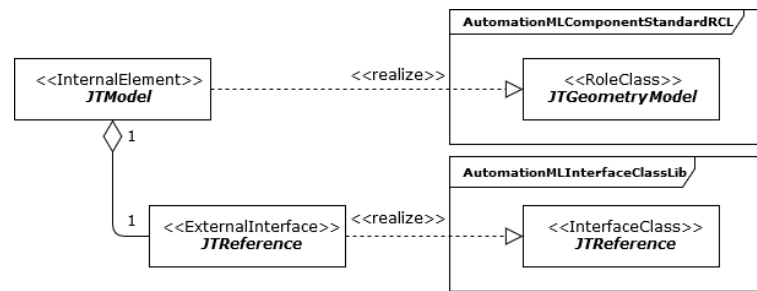


Figure 47: Example usage of the AutomationML role class “JTGeometryModel”

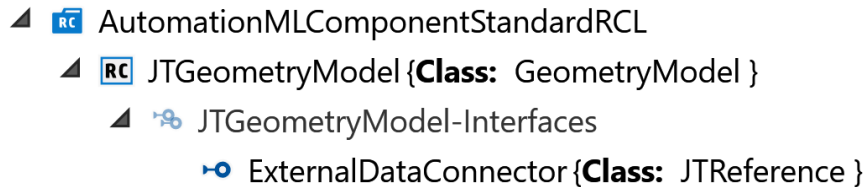


Figure 48: AutomationML Representation of “JTGeometryModel” role class

3.6.1.3 2D Model

For the integration of 2D geometry models, the following rules shall apply:

- 2D geometry model of an automation component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component as AutomationML Component.
- The InternalElement shall reference the role class “2DGeometryModel” of the role class lib “AutomationMLComponentBaseRCL” or a derived role class.
- To refer the model an ExternalInterface with the interface class “Reference2D” of the interface class lib “AutomationMLComponentBaseICL” shall be used.
- If the InternalElement represent a coordinate system that shall be published, the InternalElement shall additionally reference the role class “Frame” from the role class library “AutomationMLBaseRoleClassLib”.
- If the InternalElement defines an offset in position to a direct parent 2D geometry the attribute “Frame” with the appropriate values shall be added to that Internal Element the value for z and rz shall be “0”.

Note 1: If the attribute “Frame” is missing all sub attributes x, y, z, rx, ry, rz shall be assumed to be 0, see [WP-Part3:2017].

Note2: The translation of the InternalElement by the attribute “Frame” relates to the parent InternalElement, SystemUnitClass, InstanceHierarchy or SystemUnitClassLibrary.

Note 3: The elements InstanceHierarchy and SystemUnitClassLib specify a three dimensional, orthogonal, right-handed coordinate system with standard basis. The positive z axis is considered upward, the positive x direction defines the right axis and the negative y direction defines the forward axis.

Figure 49 depicts the general inheritance structure of the used role classes “2DGeometryModel”. Additional Figure 50 show the role class as object tree.

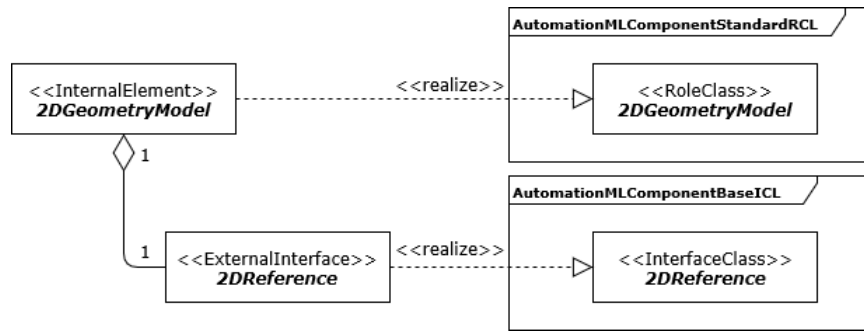


Figure 49: Example usage of the AutomationML role class “2DGeometryModel”

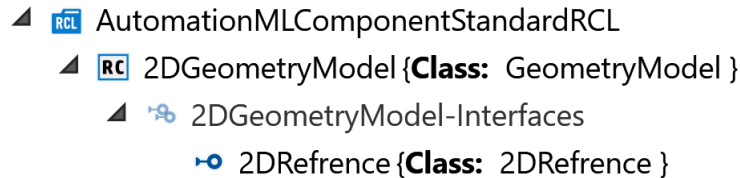


Figure 50: AutomationML Representation of “2DGeometryModel” role class

3.6.2 Kinematics

AutomationML Components might have parts that move or can be moved kinematically. The modelling of these kinematic definitions and the attachment points to the kinematics of other components is described in this chapter. Usually kinematics is closely related to geometry, but the concept shown here also allows a separate modelling of kinematics even when the geometry is not known or used to that point in time.

One model aspect of automation components and systems is the kinematic model.

For modelling a kinematic model of an automation component, the following rules shall apply:

- The base format for modelling and exchanging kinematic models shall be COLLADA.
 - Note 1: see 3.6.1.1 for COLLADA specific modelling
- Kinematic model of an automation component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component as AutomationML Component.
- The InternalElement shall reference the role class “KinematicModel” of the role class lib “AutomationMLComponentBaseRCL” or a derived role class.
 - Note 2: Each automation component may have zero or more kinematic models.
 - Note 3: The kinematic model of a component itself may consist of 1 or more kinematic sub models.
- If the complete kinematic model is stored in separated files, each file shall have a separate reference.
 - Note 4: If the files are linked within their specification scope one reference to the root of the kinematic model shall be sufficient.

Figure 51 depicts the general inheritance structure of the used role classes to integrate kinematic models. Figure 52 shows the role class as object tree.

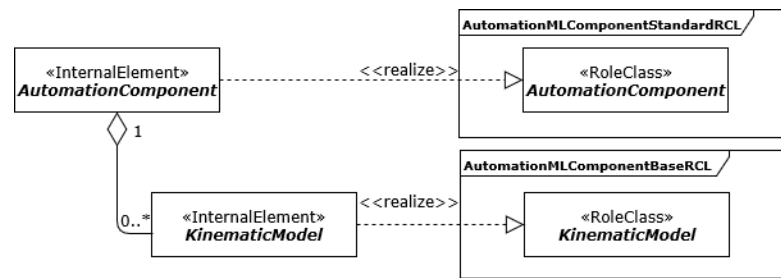


Figure 51: Example usage of the AutomationML role class “Kinematic” role class

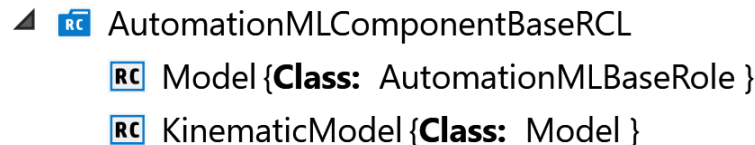


Figure 52: AutomationML Representation of “KinematicModel” role class

3.6.2.1 Collada Kinematic Model

For the integration of COLLADA kinematic models the following rules shall apply:

- All specifications form [WP-Part3:2017] shall be applied.
- A COLLADA kinematic model of an automation component shall be attached to an InternalElement, which shall be a child or sub child of the AutomationML element representing the automation component as AutomationML Component.
- The Internal Element shall reference the role class “COLLADAKinematicModel” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class
- If the Internal Element represent a coordinate system that shall be published, the Internal Element shall additionally reference the role class “Frame” from “AutomationMLBaseRoleClassLib”.
- To refer the kinematic model in a COLLADA document an ExternalInterface with the interface class “COLLADAinterface” shall be used that points to a COLLADA element “kinematics_scene” of a COLLADA document

Note 1: If one COLLADA document contains geometry, kinematics and the binding of both, the importer tool has to bind the COLLADAGeometryModel and the COLLADAKinematicModel by matching over the same file name and the bind_kinematics_model of the COLLADA kinematics_scene.

- The root COLLADA kinematic model shall be an explicit COLLADA kinematic model by setting the attribute refType of the ExternalInterface to “explicit”.

Note 2: An AutomationML Component might have zero or more root COLLADA kinematic models.

Note 3: A explicit COLLADA kinematic model shall not be a child or sub child of an explicit COLLADA kinematic model.

Note 4: An explicit COLLADA kinematic model usually references the whole kinematics of an object

- The root of COLLADA kinematic model might have zero or more “implicit” COLLADA kinematic models, COLLADA kinematic attachments or COLLADA kinematic joints as a child or a sub child.

Note 5: An implicit COLLADA kinematic model usually references a detail of the explicit COLLADA kinematic model.

Kinematic Joint

- A COLLADA Kinematic Joint is an InternalElement that shall reference the role class “COLLADAKinematicJoint” that is derived from the role class “COLLADAKinematicModel” from AutomationMLComponentStandardRCL

Note 6: A COLLADA kinematic joint represents a joint of the COLLADA model that is of interest in the CAEX part.

- The InternalElement of a COLLADA kinematic joint shall have one ExternalInterface with the InterfaceClass “COLLADAInterface” referencing the joint in the COLLADA file and one ExternalInterface with the InterfaceClass “JointInterface” of AutomationMLInterfaceClassLib

Note 7: The ExternalInterface “COLLADAInterface” is inherited from “COLLADAKinematicModel”.

Note 8: A Joint Interface can be connected by an InternalLink with other objects in order to represent a relation. E.G. If a joint interface is connected by an internal link to an output of a behaviour model, the value of the joint shall be set to the value of the behaviour model output value.

Note 9: The direction of the InternalLink defines the direction of the coupling of the Internal Link's elements. For a unidirectional coupling two InternalLinks in opposite direction are needed.

Kinematic Attachment

- A COLLADA Kinematic Attachment is an InternalElement that shall reference the role class “COLLADAKinematicAttachment” that is derived from the role class COLLADAKinematicModel from “AutomationMLComponentStandardRCL

Note 10: A COLLADA kinematic attachment represents a connectable coordinate system

Note 11: A COLLADA kinematic attachment refers to the next parent COLLADA kinematic model

- The InternalElement of a COLLADA kinematic attachment shall have one ExternalInterface with the InterfaceClass “COLLADAInterface” referencing the COLLADA kinematics_scene node and one ExternalInterface with the InterfaceClass “AttachmentInterface” of “AutomationMLInterfaceClassLib”.

Note 12: An AttachmentInterface can be connected with an InternalLink with other kinematic objects in order to create a kinematic coupling.

Note 13: The direction of the InternalLink defines the direction of the coupling of the Internal Link's elements. The element on RefPartnerSideA moves the element on RefPartnerSideB, that means that a unidirectional connection with two InternalLinks is needed for a fixed kinematic coupling.

If the COLLADA kinematic model belongs to a composition of automation components or an automation system and is not assigned to the root element of the AutomationML Component or AutomationML Composite Component, the value “implicit” of the attribute “refType” of the interface shall be allowed.

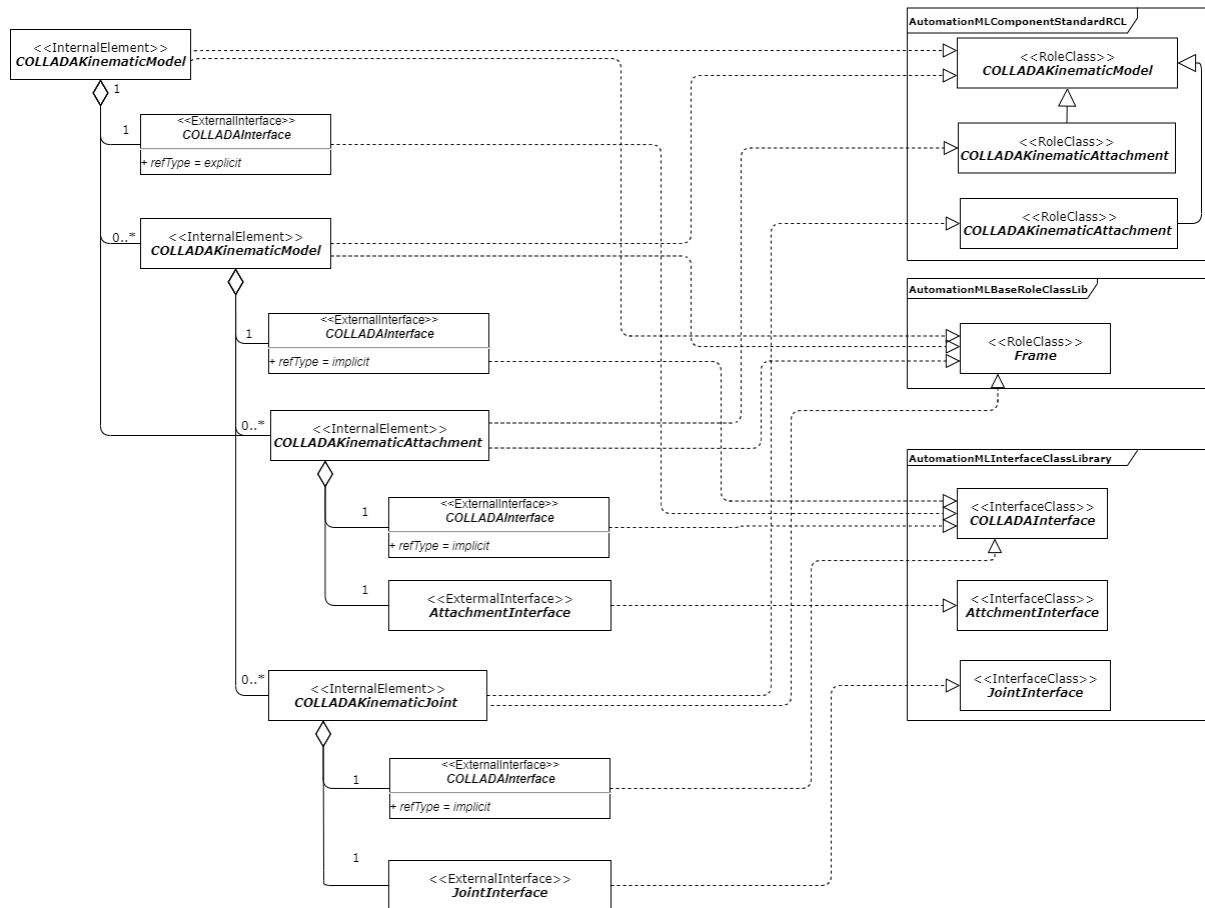


Figure 53: Example usage of AutomationML role classes to integrate an AutomationML “COLLADAKinematicModel”

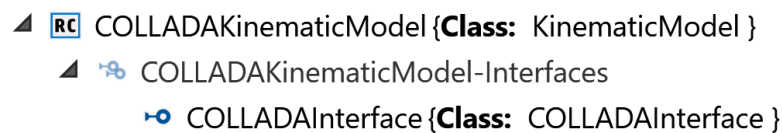


Figure 54: AutomationML Representation of “COLLADAKinematicModel” role class

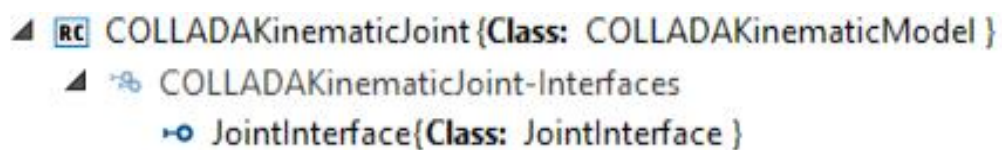


Figure 55: AutomationML Representation of “COLLADAKinematicJoint” role class

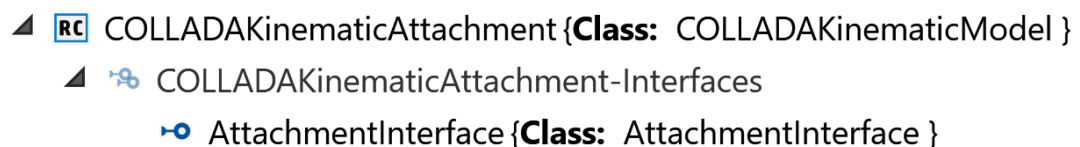


Figure 56: AutomationML Representation of “COLLADAKinematicAttachment” role class

- An COLLADAKinematicAttachment might have multiple connections to other interfaces with InternalLinks to model relations between the kinematic model and other models, objects or connectors

Note 15: The direction of the InternalLink defines the direction of the coupling of the Internal Link's elements. The element on RefPartnerSideA moves the element on RefPartnerSideB, that means that a unidirectional connection with two InternalLinks is needed for a fixed kinematic coupling.

- If an InternalLink is connected to an ExternalInterface of an element with the role class “Connector” or any derived, the InternalLink is handover to the ExternalInterface

Note 16: RefPartnerSideA on the AttachmentInterface is implicitly connected to RefPartnerSideA to any InternalLink on the ExternalInterface of the Connector

Note 17: This can be used for coupling of e.g. mechanical connectors to the internal kinematics of an AutomationML Component

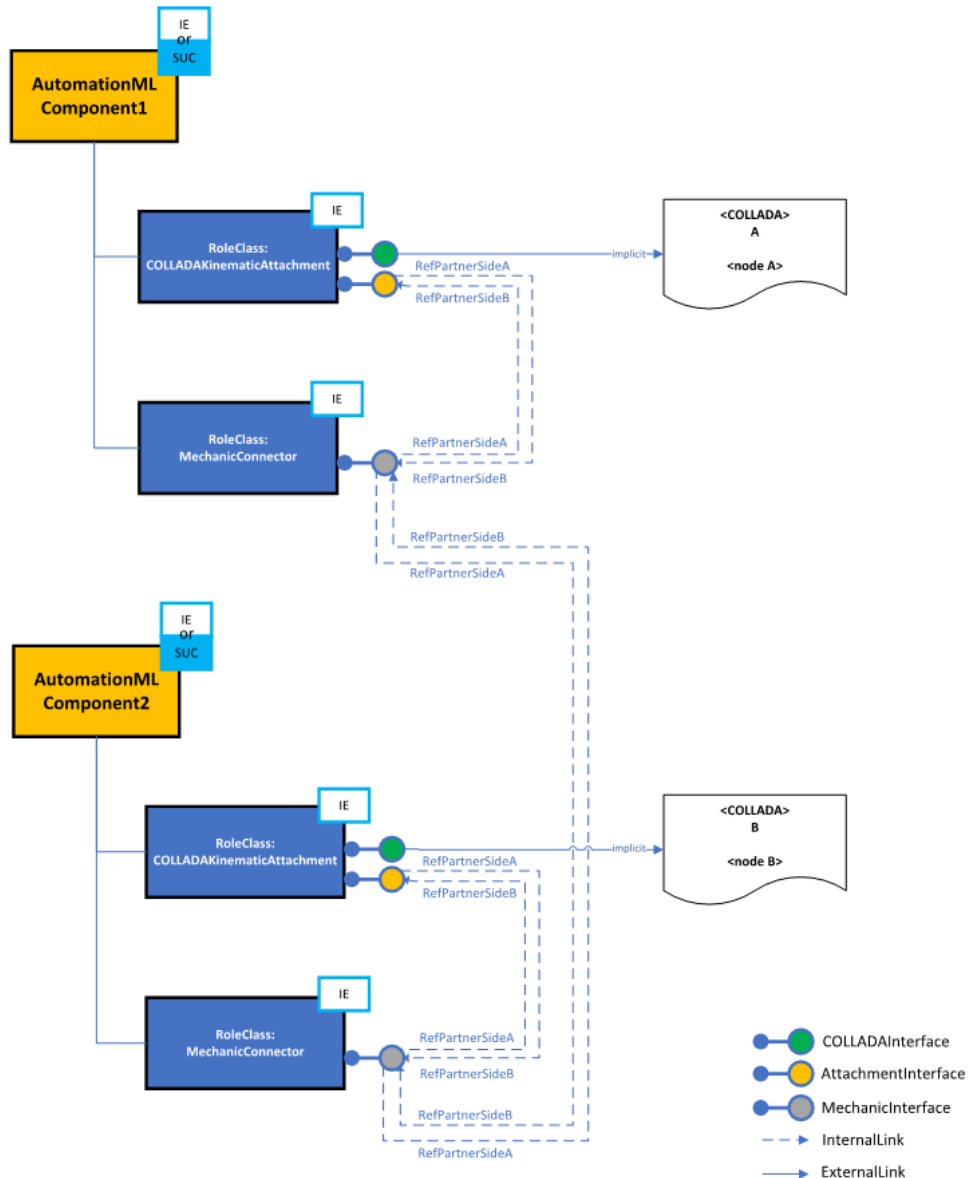


Figure 46: Fixed kinematic coupling of two AutomationML Components using Attachments over MechanicConnectors as a proxy

Example of COLLADA kinematic joint connection with FMI behaviour interface.

A COLLADA kinematic joint might have multiple connections to other Interfaces with InternalLinks to model relations between the kinematic model and other models, objects or connectors. A typical

connection for the realization of virtual commissioning is to connect the kinematic CAD model to a simulated behaviour model.

Figure 56 shows the example of a modelling of COLLADA kinematics connected with a FMLogic: The joint interface that points to the variable of the joint inside the COLLADA file can be connected to the corresponding variable interface of the FMI, that represents the certain value of the joint position.

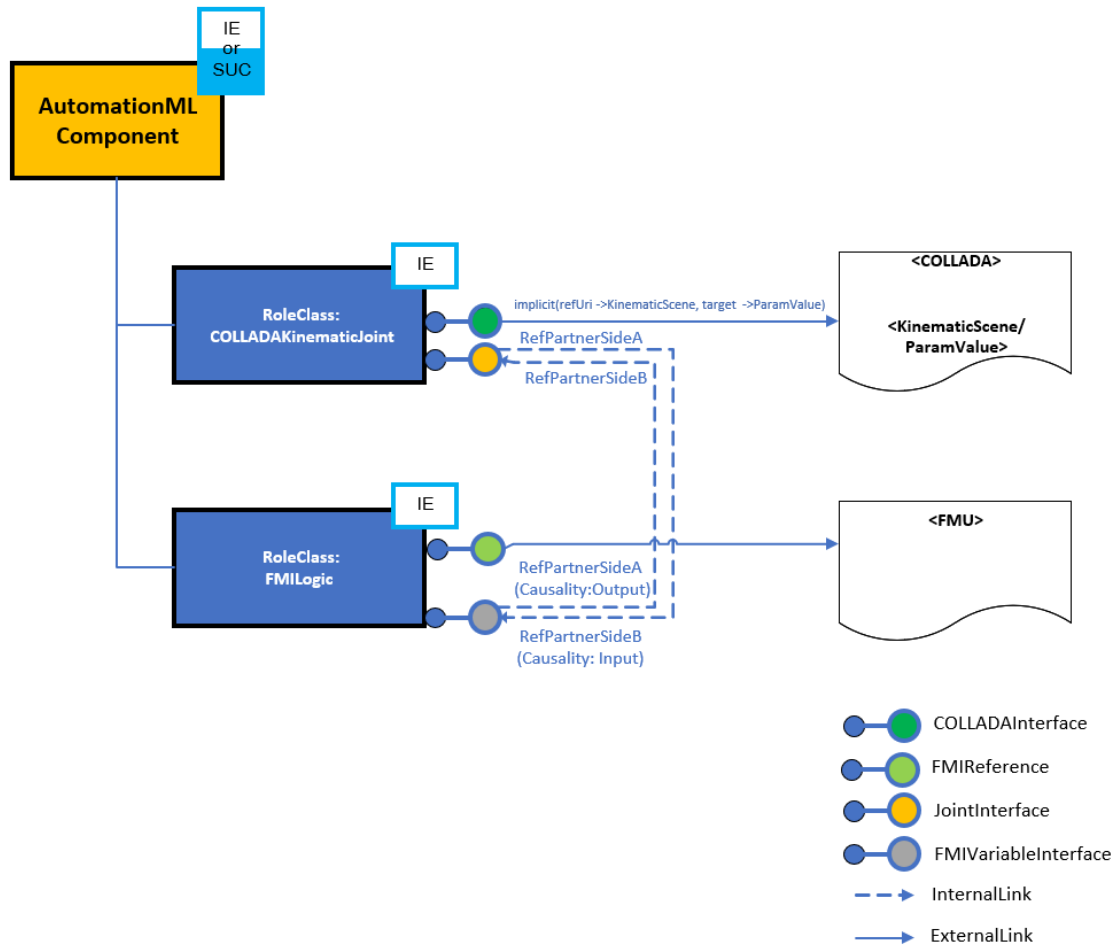


Figure 57: Example of Connection between COLLADAKinematicJoint und FMLogic

3.7 Graphic Representation

One aspect of automation components and systems is the graphic representation in different authoring and engineering tools. To reference the graphic representation, model an InternalInterface with the role class “GraphicRepresentationReference” or a derived role class shall be used. Within the description of automation components and systems three types of graphical representations are distinct:

- Symbols
- Component pictures
- Icons

The following rules shall apply:

- Graphical Representations for an automation component shall be attached as an InternalElement, which shall be a child or sub child element of the InternalElement representing the automation component.
- For a symbol, the InternalElement shall reference the role class “Symbol” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- For a picture, the InternalElement shall reference the role class “ComponentPicture” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- For an Icon, the InternalElement shall reference the role class “Icon” of the role class lib “AutomationMLComponentBaseRCL” or a derived role class.

Figure 58 depicts the general inheritance structure of the used role classes to graphical representation.

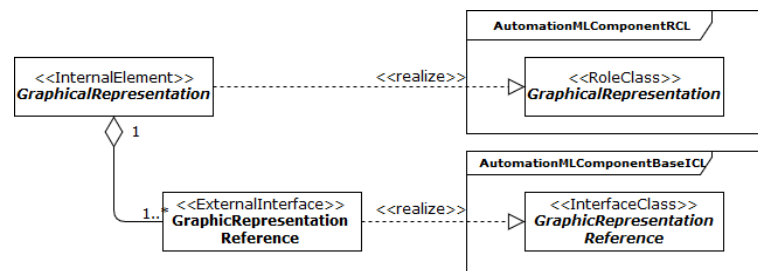


Figure 58: Example usage of the AutomationML role class “GraphicRepresentation”

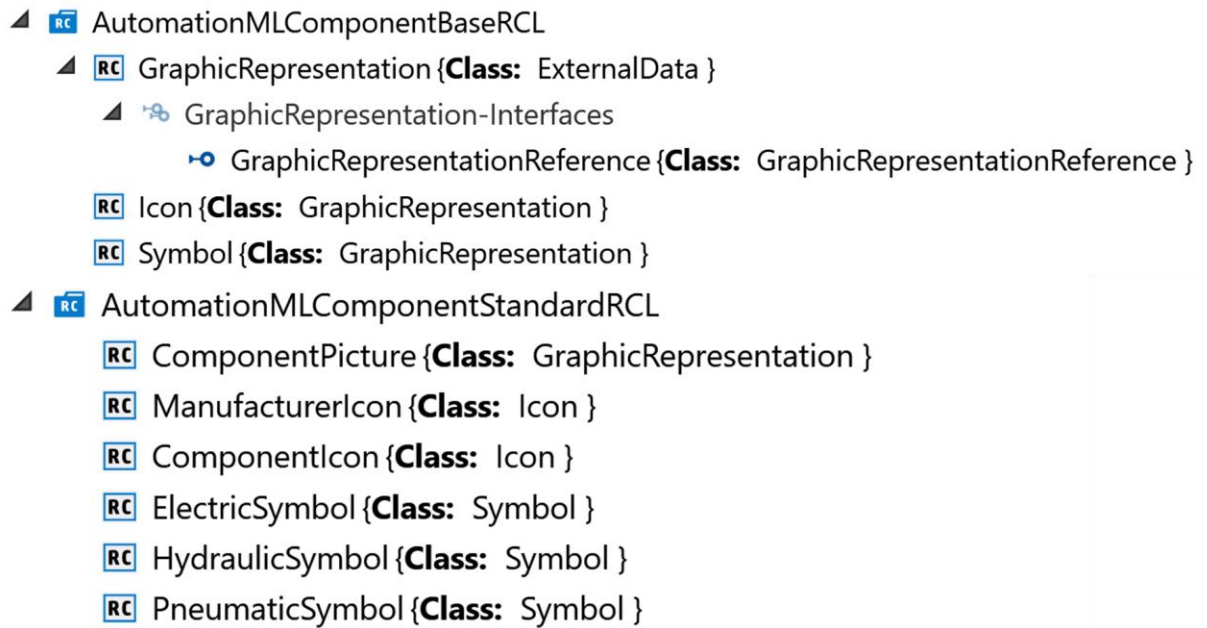


Figure 59: AutomationML Representation of “GraphicalRepresentation” role class

3.8 Documentation

The integration of documentation information for automation components and systems is an important aspect. To reference documentation information the provisions defined in [BPR-EDRef:2016] shall be followed.

The following rules shall apply additionally:

- Documentation for an automation component shall be attached as an InternalElement, which shall be a child or sub child element of the InternalElement representing the automation component as AutomationML Component.
- For documentation items, the InternalElement shall reference the role class “Documentation” of the role class lib “AutomationMLComponentBaseRCL” or a derived role class.

Figure 60 depicts the general inheritance structure of the used role classes to graphical representation.

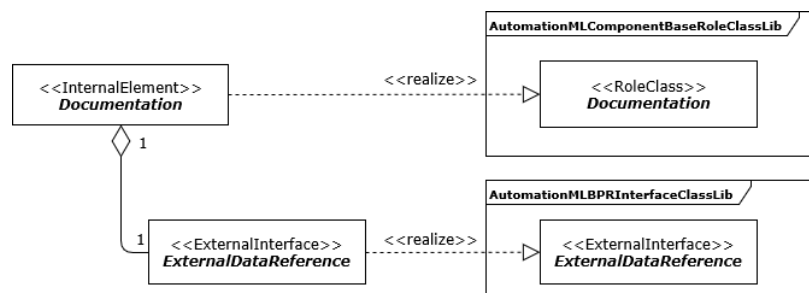


Figure 60: Example usage of the AutomationML role class “Documentation”

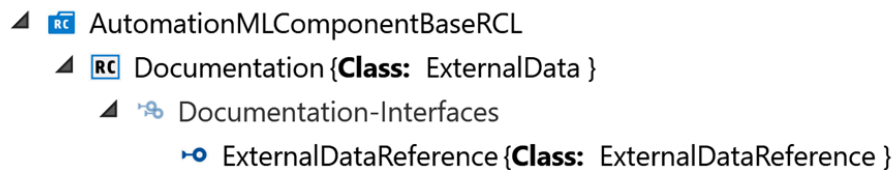


Figure 61: AutomationML Representation of “Documentation” role class

3.9 Certificates

The integration of certificate information for AutomationML Component and Composite Components is an important aspect. To refer certificate information the provisions defined in [BPR-EDRef:2016] shall be followed.

The following rules shall apply additionally:

- Certificates for an automation component shall be attached as an InternalElement, which shall be a child or sub child element of the InternalElement representing the automation component as AutomationML Component.
- For certificate items, the InternalElement shall reference the role class “Certificate” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.

Figure 60 depicts the general inheritance structure of the used role classes to graphical representation.

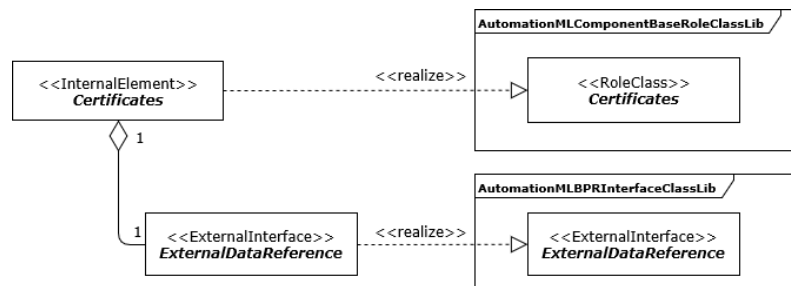


Figure 62: Example usage of the AutomationML role class “Certificate”

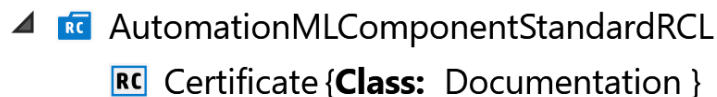


Figure 63: AutomationML Representation of “Certificate” role class

3.10 Additional Device Description

The communication interface of a modern field device is usually described by a technology related device description file such as GSDML, FDCML, ESI, CSP+, IODD and others.

The integration of such existing device descriptions information for automation components and systems is an important aspect and the following rules shall apply:

- An Additional Device Description for an automation component shall be attached as an InternalElement referencing a role class derived from the abstract role class “AdditionalDeviceDescription”, which shall be a child or sub child element of the InternalElement representing the automation component as AutomationML Component.
- To reference to the external file itself, the refURI attribute from interface class “DeviceDescriptionReference” shall be used. Same as with the role class above, the interface class is a basic abstract interface which shall not be used directly. A derived interface class referring to the specific type of the device description file shall be used.
- An automation component can contain multiple InternalElements that represent a Device Description, each of them having exactly one ExternalInterface to refer to a file. The following scenarios for multiple Additional Device Descriptions for one automation component are possible:
 - a) For a field device different versions of the device description files are available
 - b) Different versions of the device description language/xsd schema are existing and for each of them a device description file is provided for the field device
 - c) A gateway module has communication interfaces for different networks and therefore a device description file for each of them

- d) A combination of the above
- The attribute “SpecVersion” in the role class shall be used to define the version of the device description language/xsd schema, respective the technology DD specification.
- The attribute “Version” in the interface class defines the version of the actual device description file that is referenced.
- The attribute “DocLang” in the role class shall be used to specify the language of the referenced device description file according to [RFC5646:2009].

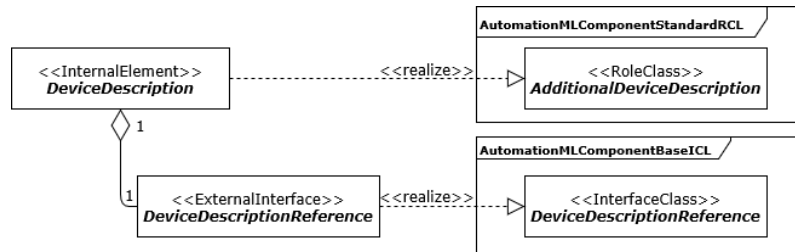


Figure 64: Example usage of the AutomationML role class “AdditionalDeviceDescription”

AutomationMLComponentBaseRCL

AdditionalDeviceDescription {Class: ExternalData }

AdditionalDeviceDescription-Interfaces

DeviceDescriptionReference {Class: DeviceDescriptionReference }

Figure 65: AutomationML Representation “AdditionalDeviceDescription” role class

3.11 Maintenance Description

For an automatic generation of a production line maintenance plan, it is necessary to have a computer readable component maintenance description in a computer readable way.

A maintenance description should be implemented as one or more MaintenanceDescriptionGroup, which describes the main topic of the maintenance. The details of the maintenance description are described as a MaintenanceDescriptionItem which is always a child of a MaintenanceGroup.

The following rules shall apply additionally:

- A maintenance description shall be modeled as an InternalElement with the role class “MaintenanceDescriptionGroup” or any derived role class as a child of an AutomationML Component root element.
- A maintenance item shall be modeled as an InternalElement with the role class “MaintenanceDescriptionItem” as a child of an MaintenanceDescriptionGroup element.

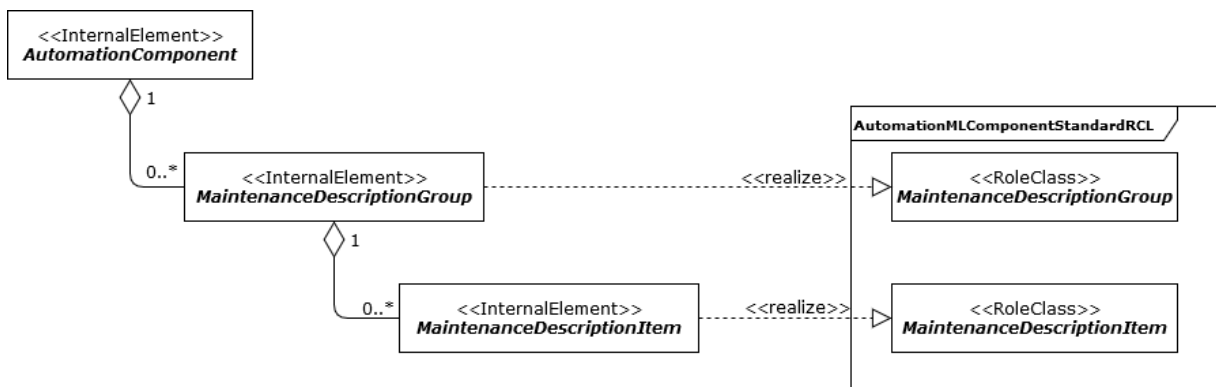


Figure 66 depicts the general inheritance structure of the used role classes to maintenance description representation.

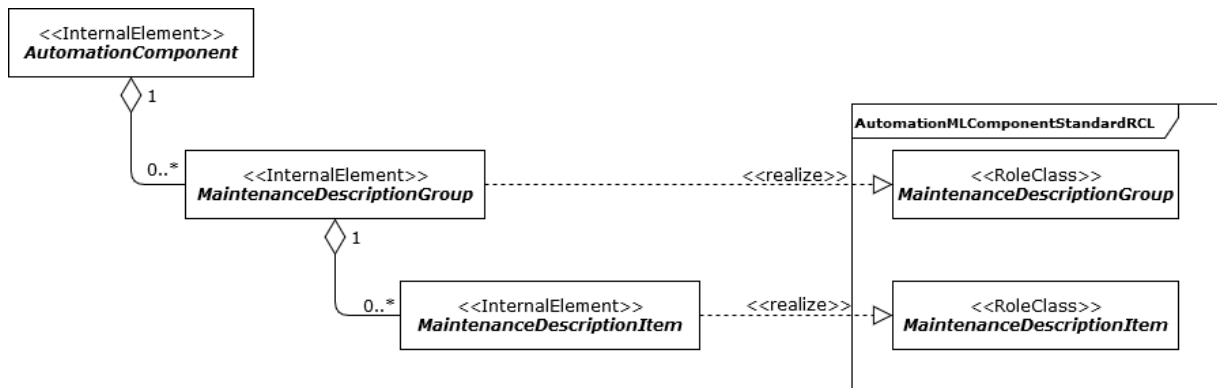


Figure 66: Example usage of the AutomationML role classes “MaintenanceDescriptionItem” and “MaintenanceDescriptionGroup”

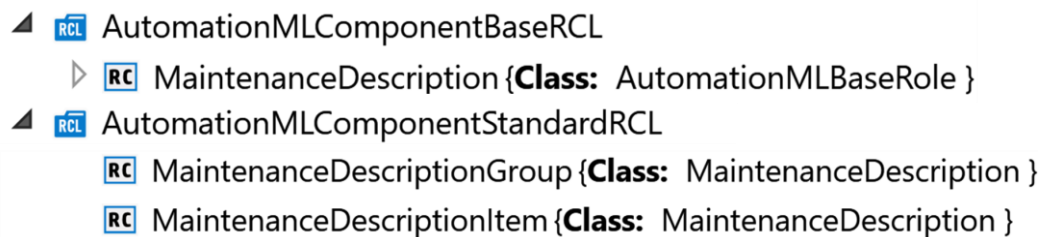


Figure 67: AutomationML Representation of “MaintenanceDescription” role classes

3.12 Skill Description for AutomationML Components

One model aspect of automation components and systems is related to their provided skills. Platform4.0 defines skill as “Potential of an Industrie 4.0 component to achieve an effect within a domain”. Additional notes as follows are given on the definition of skill allowing the interpretation of skills in different ways:

- Skills can be described as the sum of all properties (see IEC61360).
- The comparison between requirements and assurances is realized via properties of the skill.
- Skill can be orchestrated and hierarchically structured.
- Capability is often used as synonym to skill.
- Skills can be made executable via services
- The impact manifests in a measurable effect within the physical world.

This document adopts the definition of skill from platform 4.0 [Platform4.0 Glossary]

Accordingly an internal element with the role class “SkillModel” or a derived role class shall be used in order to refer a skill of the automation component.

An InternalElement with the role class “SkillModel” or a derived role class shall be used in order to refer a skill of the automation component.

Within this version of the document basic concepts of modelling skills for automation components and systems are defined. **A detail definition how to model skills in AutomationML Components will be part of Version 2 of the document.**

Following basic provisions shall be applied in order to integrate model skills of automation components into an AutomationML file:

- Skills model information of an AutomationML Component shall be attached to an InternalElement.

- The InternalElement carrying skill information shall reference the role class “SkillModel” of the role class library “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement carrying the skill logic information shall reference the role class “SkillLogicModel” of the role class library “AutomationMLComponentStandardRCL” and one of the role classes “PLCLogic”, “AMLLLogic” or “FMILogic” or a derived role class of the role class library “AutomationMLComponentBaseRCL”.
- The InternalElement describing the connections of skills shall reference the role class “SkillConnector” or a derived role class library “AutomationMLComponentStandardRCL”.

Note: The connection could be a connection within an AutomationML Component between different models or between skills of different AutomationML Components process or products.

- The InternalElement carrying the skill logic and the InternalElement describing the connection shall be child elements of an InternalElement with the role class “SkillModel”.
- The InternalElement carrying the RoleClasses for specific models defined in chapter 4.1.3.6 shall be used, if additional properties of the component is integrated within skill.

Figure 68 depicts the general AutomationML representation of skill description as UML structure. Additional Figure 69 shows the role class as object tree.

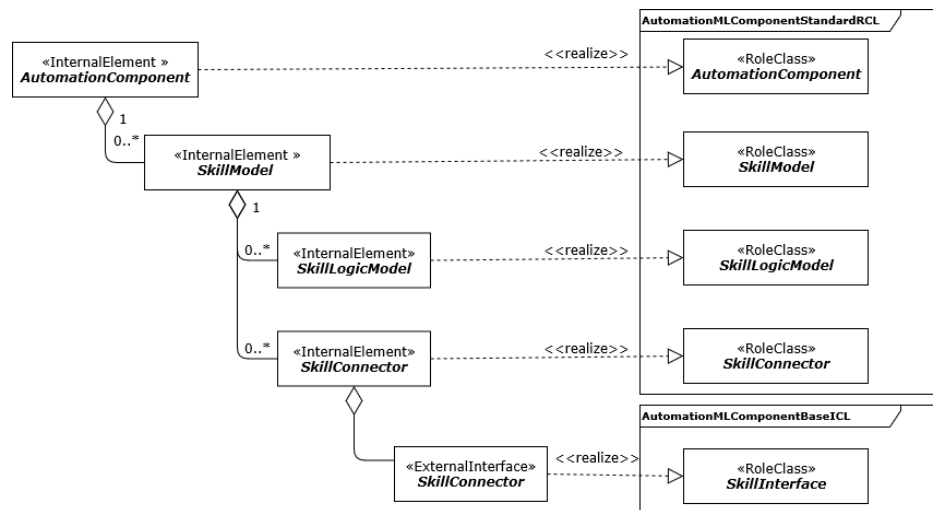


Figure 68: General AutomationML Representation of Skill Description

- ▲ **ICL** AutomationMLComponentBaseICL
 - IC** SkillInterface {**Class:** AutomationMLBaseInterface }
- ▲ **RCL** AutomationMLComponentStandardRCL
 - RC** SkillModel {**Class:** Model }
 - RC** SkillConnector {**Class:** Connector }
 - RC** SkillLogicModel {**Class:** LogicModel }

Figure 69: AutomationML Representation of skill model role and interface classes

3.13 Connector for AutomationML Components

3.13.1 General

Automation Systems and Components are resources of an automated system that might have logic or physical connections to other systems or components. These connections shall be described with a separate AutomationML object that has a well-defined role class. This chapter describes how to model theses connectors and which role classes an interface types are defined to model them.

A set of role classes and interface classes for semantical tagging of AutomationML objects as connectors is defined. Figure 70 and Figure 71 depict these classes in an AutomationML tree view.

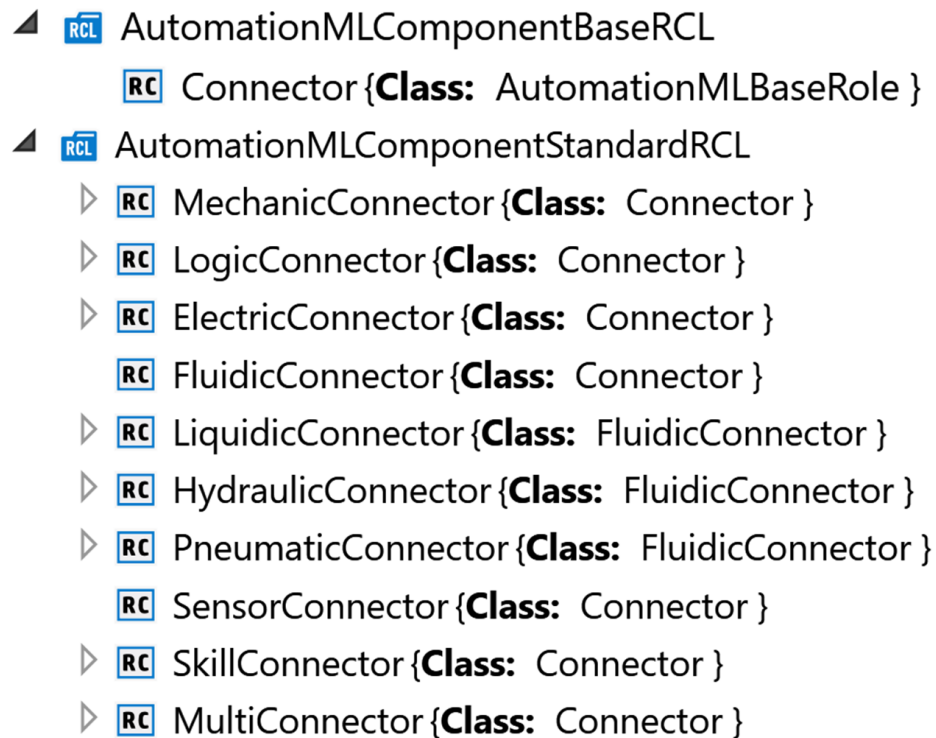


Figure 70: Role classes AutomationML Component connector definition

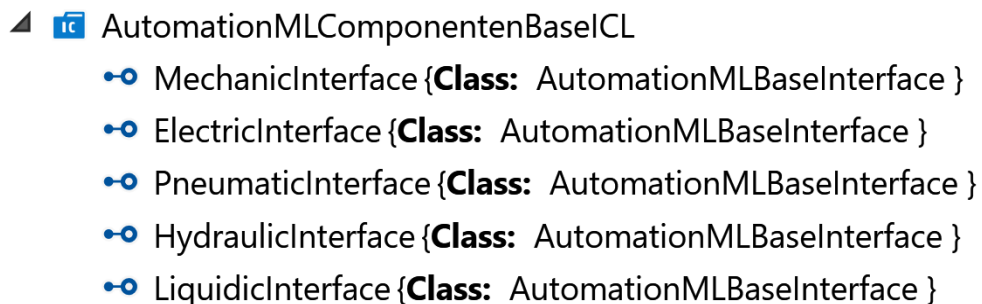


Figure 71: Interface classes for AutomationML Component connector definition

For the use of connectors following basic rules shall apply:

- An AutomationML Component connector shall be modeled as an InternalElement with the role class "Connector" or any derived role class as a child of an AutomationML Component root element.
- Any InternalElement defined as connector shall have at least one ExternalInterface.

Figure 72 depicts the example usage of AutomationML Component connector to describe AutomationML Component connectors.

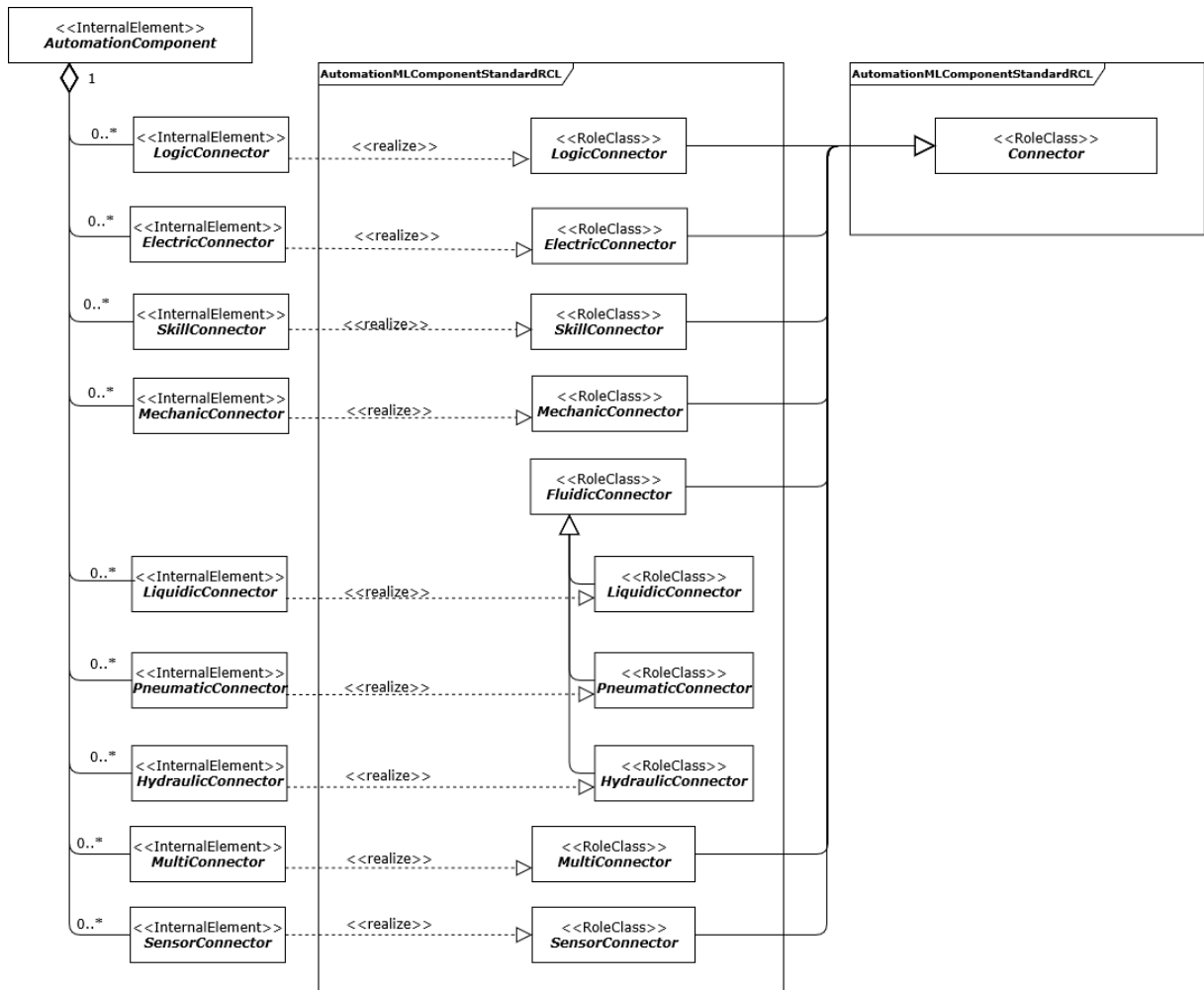


Figure 72: Representation of functions as AutomationML entities (UML diagram)

A detailed description for the use of AutomationML Component connector to describe the interconnection of AutomationML Components can be found in chapter 3.13.

3.13.2 Mechanic Connector

The mechanic connector is the representation of a mechanical fastening interface of an automation component.

It mechanically affixes or fastens one or more objects in means of a non-permanent joint that can be removed or dismantled without damaging the joining components. The joint can transfer energy (e.g. motion, force), that can be defined in detail, if a behaviour model is connected the mechanic connector.

For the description of a mechanic connector, the following provisions shall apply:

- A mechanic connector of an automation component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component.
- The InternalElement shall reference the role class “MechanicConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The mechanical interface shall be modelled as an “MechanicInterface” from AutomationMLComponentBaseICL or a derived Interface.

Figure 73 depicts the general inheritance structure of the used role classes define a mechanic connector. Figure 74 shows the role class as object tree.

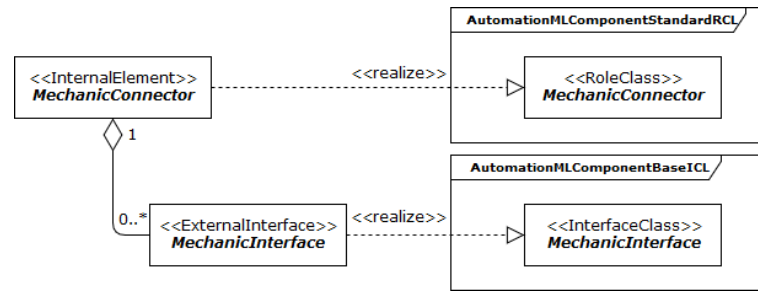


Figure 73: Example usage of an AutomationML “MechanicConnector” role class

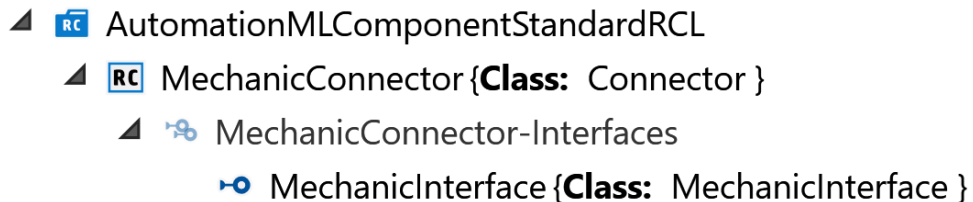


Figure 74: AutomationML representation of “MechanicConnector” role class

3.13.3 Logic Connector

A logic connector is an interface that allows to connect virtual objects such as signal or process variables to each other or to their physical representation such as an electrical or mechanical interface.

For the description of logic connectors, the following provisions shall apply:

- A logic connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component.
- The InternalElement shall reference the role class “LogicConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.

Figure 75 depicts the general inheritance structure of the used role classes to define a logic connector. Figure 76 shows the role class as object tree.

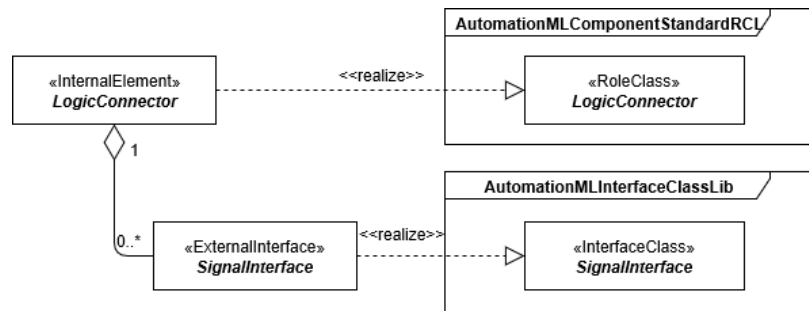


Figure 75: Example usage of an AutomationML “LogicConnector” with one “SignalInterface”

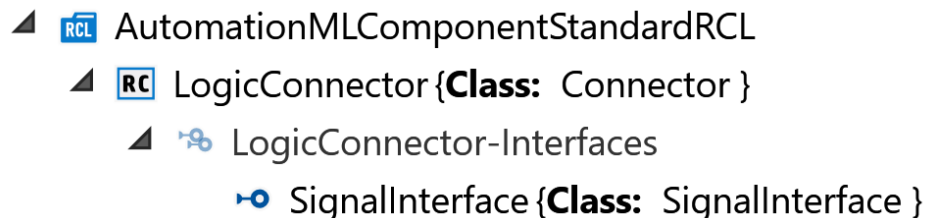


Figure 76: AutomationML Representation of “LogicConnector” role class

3.13.4 Electric Connector

3.13.4.1 General

Electric connectors are essential part of almost every automation component. They are used to supply energy to a device or to connect the device for engineering, configuration or data exchange with the automation system. The industrial evolution has developed a variety of different electric connector types, examples are shown in Figure 77.



Figure 77: Examples of electric connectors

Electric connectors vary from mechanical connectors without any further function, e.g. a M12 or RJ45 plugs (Figure 78 a) to complex connectors with computational power, e.g. M12 or RJ45 Ethernet. For flexible connectivity, modern automation devices may contain one or multiple electric connectors (see Figure 78 b) used for a variety of purposes. Electric connectors may also be of non-physical nature, e.g. a WIFI connection. This clause defines modelling principles for electric connectors.



Figure 78: Electric connectors used in a cable or in an automation devices

- a) Example M12 to M12 cable with a male plug and a female socket connector with inner pins (interfaces)
- b) Example automation component with a variety of electric connectors with interfaces [source: BALLUFF]

The first general principle is the separation between *connector* and *interface*. The connector is the overall mechanical entity, while the electric interface is the concrete electrical pin that transfers electric current. Pins form the end of electric wires and electric connectors allow to connect set of wires.

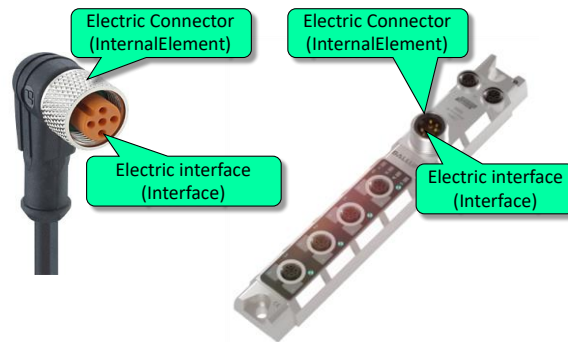


Figure 79: Connector versus Interface

The second principle is the connector application. The following example illustrates this: an M12 plug may be used either as energy supply or as Ethernet port. An M12 plug at the end of a cable does not know its application, it might be plugged into an energy connector or an Ethernet port. An electric connector is, by design, application independent. The application modelling requires additional modelling effort, usually a parent InternalElement that models the application. The AutomationML modelling of an electric connector focus on the application independent connector only. The application of an electric connector is usually implemented by a logic circuit that is e.g. developed to support Ethernet. This logic makes an M12 connector an ethernet connector. The circuit is not part of the present model.

Finally, an electric connector is an electro-mechanical provision used to join electrical terminations and to create an electrical circuit. Electric connectors often consist of mechanical interfaces as male-ended plugs and jacks or sockets (female-ended) that usually contain multiple electrical pins, and of logic circuits that implement functionality for sending, receiving or processing electrical signals. The connection may be temporary or serve as a permanent electrical joint between devices.

An electric interface is an electro-mechanical provision modelling the connectable items of an electric connector. This may be single pins, or mechanical plugs or sockets containing multiple pins with dedicated geometry. Examples for electric interfaces are M12, RJ45, Mini 7/8 as shown in Figure 77, or RJ232, USB 3, Thunderbolt and many more (see Figure 77).

3.13.4.2 General modelling provisions for electric connectors

The Automation Component standard library provides two base classes for modelling electric connectors: the role class “ElectricConnector” and the interface class “ElectricInterface” (see Figure 80 and Figure 81). An electric connector can contain one or more electric interfaces, whereas the electric interfaces correspond to the pins of the connector. The ExternalInterface Figure 77 depicts the general inheritance structure of the used role classes to define an electric connector. Figure 78 shows the role class as object tree.

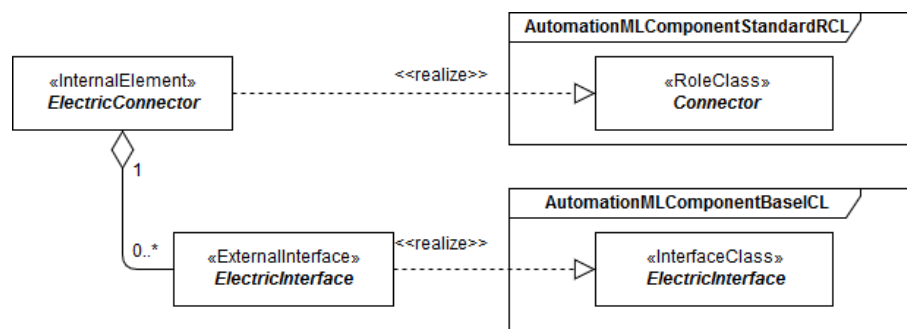


Figure 80: Example usage of an AutomationML “ElectricConnector” with one ElectricInterface

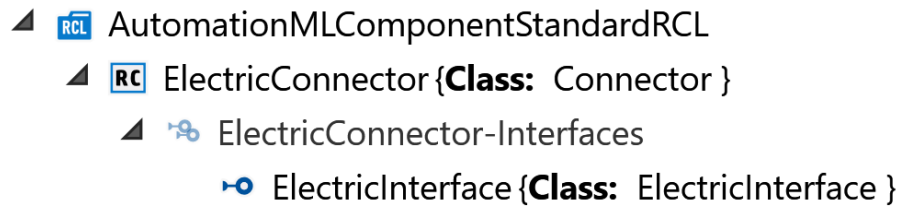


Figure 81: AutomationML Representation of “ElectricConnector” role class

For the AutomationML modelling of electric connectors, the following provisions apply:

- An electric connector of an automation component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the automation component as AutomationML Component.
- The InternalElement shall directly or indirectly reference the role class “ElectricConnector” of the role class lib “AutomationMLComponentStandardRCL”.
- The electric interface of the connector shall be modelled as CAEX ExternalInterface.
- If the connector has pins, the electric interface shall be modelled as CAEX ExternalInterface of the InternalElement which represents the connector.
- If the connector has nested connectors, the child connectors shall be modelled as child InternalElements of the parent connector, which directly or indirectly references the RoleClass “ElectricConnector”. Then, the sub interfaces of the children shall be modelled as CAEX ExternalInterfaces of the children or sub-children. This allows modelling complex and nested super-connectors with inner sub connectors with arbitrary deepness. The ending leaves of this tree are electric interfaces.
- Each CAEX ExternalInterface that models an electric interface shall be directly or indirectly derived from the InterfaceClass “ElectricInterface” of the interface class library “AutomationMLComponent-BaseICL”.
- An electric connector shall describe its physical characteristics, consisting of the physical connector type with its properties (i.e. M8, M12, RJ45, clamp, male, female, max voltage, max current etc.) and the underlying single electrical pins with its properties.

Note 1: Only electric interfaces can be connected to each other. It is not possible to directly connect the connectors with the present version of CAEX 2.15. The future modelling based on CAEX 3.0 allows this and is described in clause 8.4.

Note 2: To model a complex interface in CAEX 2.15 the port concept defined in [WP-Part1:2016] according to chapter A.1.2 should be used. Therefor the ElectricConnector shall reference the role class “Port” of the “AutomationMLBaseRoleClassLib” including the ExternalInterface “ConnectionPoint”. This External interface shall be used for the connection of the ElectricConnector.

3.13.5 Fluidic Connector

The Fluidic Connector is a base connector for modelling connectors of automation components that handle fluidics. For the description of fluidic connectors, the following provisions shall apply:

- A fluidic connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the Automation Component.
- The InternalElement shall reference the role class “FluidicConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.

Figure 82 depicts the usage of the role classes “FluidicConnector”. Additional shows Figure 83 the role class as object tree.

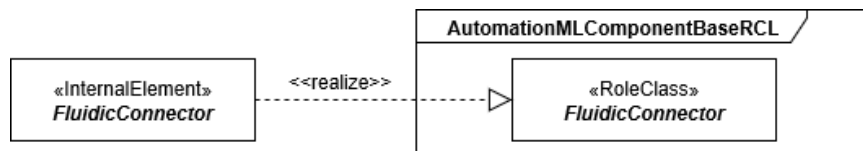


Figure 82: Example usage of an AutomationML “FluidicConnector”



Figure 83: AutomationML representation of “FluidicConnector” role class

3.13.6 Liquidic Connector

The Liquidic Connector models connectors of automation components that handle liquids. It derives from the Fluidic Connector.

- A liquidic connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the Automation Component.
- The InternalElement shall reference the role class “LiquidicConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement shall have one or more interfaces from interface class “LiquidicInterface” of “AutomationMLComponentBaseICL”

Figure 84 the usage of the role classes “LiquidicConnector”. Additional shows Figure 85 the role class as object tree.

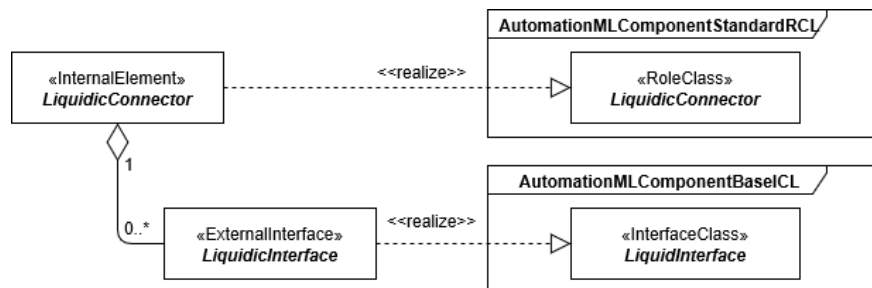


Figure 84: Example usage of an AutomationML “LiquidicConnector”

- ▲ **RCL** AutomationMLComponentStandardRCL
 - ▲ **RC** LiquidicConnector {**Class:** FluidicConnector }
 - ▲ **LI** LiquidicConnector-Interfaces
 - LiquidicInterface {**Class:** LiquidicInterface }

Figure 85: AutomationML Representation of “LiquidicConnector” role class

3.13.7 Pneumatic Connector

The Pneumatic Connector models connectors of an automation component that handle compressed air. It is derived from the Fluidic Connector.

Base for the Pneumatic Connector will be the upcoming Application Recommendation Pneumatics of the AutomationML consortium, which relies on [ISO 18582-2] standard.

For the description of pneumatic connectors, the following provisions shall apply:

- A pneumatic connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the Automation Component.
- The InternalElement shall reference the role class “PneumaticConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement shall have one or more Interfaces from interface class “PneumaticConnector” or “CondensateDrainConnector” or both of “AutomationMLComponentBaseICL”.
- The InterfaceClass PneumaticConnector has the attributes pneumatic port and connector type with all values defined by [ISO 18582-2].

Figure 86 depicts the example usage of the role class “PneumaticConnector”. Additional Figure 87 and Figure 88 show the role class and the used interface classes as object tree.

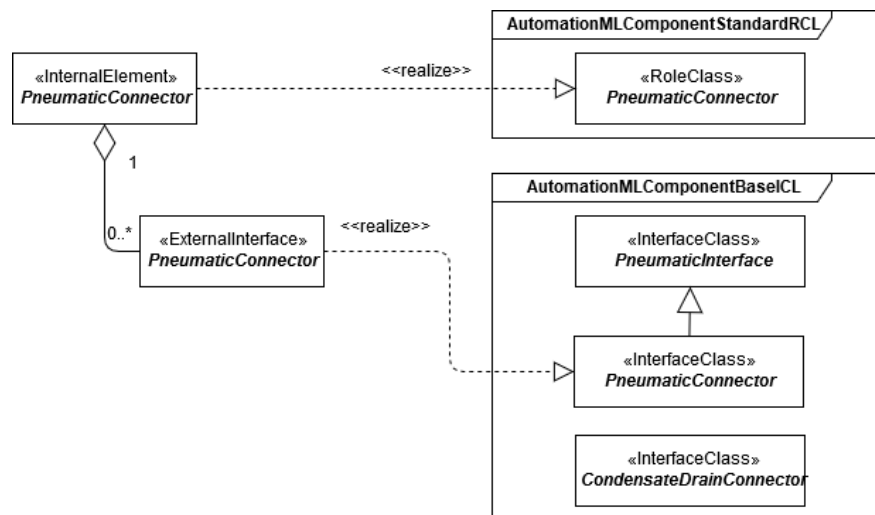


Figure 86: Example usage of an AutomationML “PneumaticConnector”

- ▲ **RCL** AutomationMLComponentStandardRCL
 - ▲ **RC** PneumaticConnector {**Class:** FluidicConnector }
 - ▲ **LI** PneumaticConnector-Interfaces
 - PneumaticInterface {**Class:** PneumaticInterface }

Figure 87: AutomationML Representation of “PneumaticConnector” role class





- ▲  AutomationMLComponentBaseICL
 -  PneumaticInterface {**Class:** AutomationMLBaseInterface }
 -  PneumaticConnector {**Class:** PneumaticInterface }
 -  CondensateDrainConnector {**Class:** AutomationMLBaseInterface }

Figure 88: AutomationML Representation of “PneumaticInterface” and “PneumaticConnector” interface class

3.13.8 Hydraulic Connector

The Hydraulic Connector models connectors of automation components that handle hydraulic fluid. It derives from the Fluidic Connector. For the description of hydraulic connectors, the following provisions shall apply:

- A hydraulic connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the Automation Component.
- The InternalElement shall reference the role class “HydraulicConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement shall have one or more Interfaces from interface class “HydraulicConnector” of “AutomationMLComponentBaseICL” or a derived interface class.

Figure 89 depicts the example usage of the role class “HydraulicConnector”. Additional shows Figure 90 the role class as object tree.

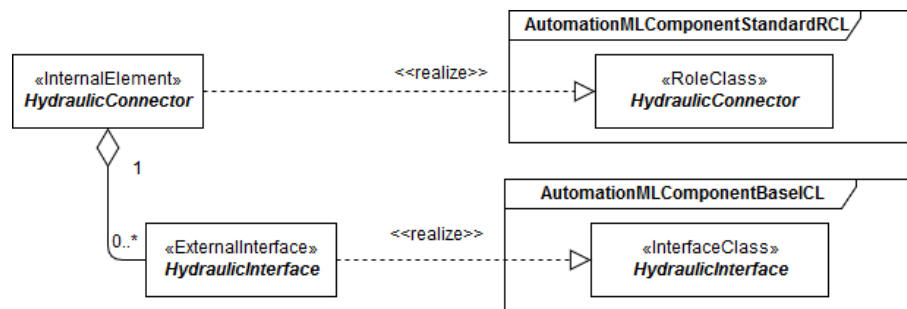


Figure 89: Example usage of an AutomationML “HydraulicConnector”





- ▲  AutomationMLComponentStandardRCL
 - ▲  HydraulicConnector {**Class:** FluidicConnector }
 - ▲  HydraulicConnector-Interfaces
 -  HydraulicInterface {**Class:** HydraulicInterface }

Figure 90: AutomationML Representation of “HydraulicConnector” role class

3.13.9 Sensor Connector

A Sensor Connector is a special case of a mechanical connector. It is the process interface to physically sense the properties of interest (i.e. a mechanical movement or object presence within a spatial detection area or mediums states like temperature or pressure). Sensors of many physical principles have such a spatial detection area with a certain geometry and detection properties, which needs to be specified both for mechanical construction and simulation environments. Properties of the model shall be geometry, direction, attenuation over distance, type of medium, etc. Other sensor types (i.e. fluidic) may use different properties. The details of sensor connectors will be described in Version 2 of this document.

3.13.10 Skill Connector

The skill connector shall be used to describe connections between different Automation Component skills.

For the description of skill connectors, the following provisions shall apply:

- A skill connector of an Automation Component shall be attached to an InternalElement, which shall be a child or sub child element of the AutomationML element representing the Automation Component. Additionally, the InternalElement shall reference the role class “SkillModel” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The parent InternalElement shall reference the role class “SkillConnector” of the role class lib “AutomationMLComponentStandardRCL” or a derived role class.
- The InternalElement shall have one or more Interfaces from interface class “SkillInterface” of AutomationMLComponentBaseICL or a derived interface class.

Figure 89 depicts the example usage of the role class “SkillConnector”. Additional shows Figure 90 the role class as object tree.

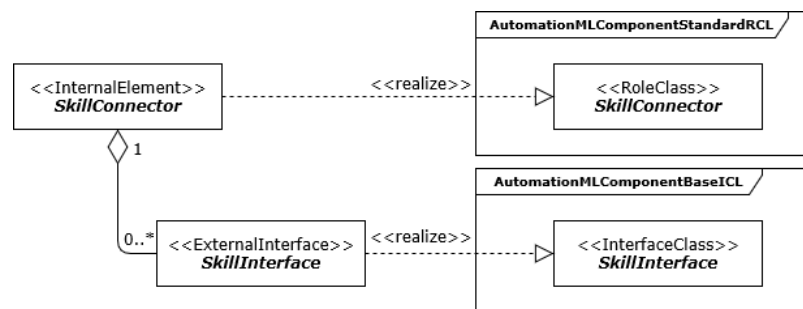


Figure 91: Example usage of an AutomationML “SkillConnector” role class

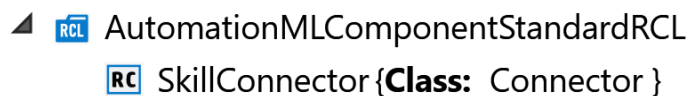


Figure 92: AutomationML Representation of “SkillConnector” role class

A detail specification of the usage of skill connectors will be described in Version 2 of this document.

3.13.11 Multi Connectors

A multi connector is a mechanical provision containing multiple of the previously described basic connector types. Such multi connector will be described in Version 2 of this document.

3.14 Connecting AutomationML Components and Composite Components

3.14.1 Relations between the contents within an AutomationML Component

The previous chapters have described the modelling of internal data models and connectors of an AutomationML Component. A major strength of AutomationML is the representation of interconnected information even if it is from other domains (interdisciplinary). For instance, AutomationML can set relations between geometry, kinematics and behaviour in order to represent a full virtual commissioning digital twin.

The relations described here are targeted to be specified in a second version of this document and are just roughly sketched and explained on examples here.

Figure 93 shows an abstract graphical visualization of the main contents of an example of an AutomationML Component. In the lower half of the graphics InternalLinks between Connectors and Models and between just Models are shown by different colored lines.

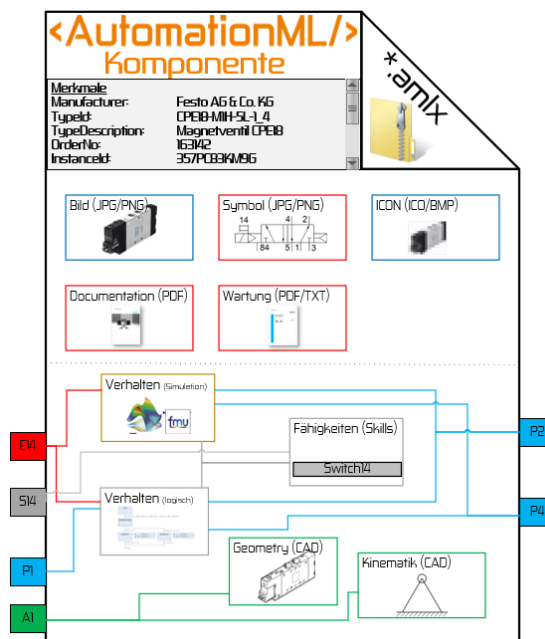


Figure 93: Graphical visualization of the main contents of an AutomationML Component.

For the modelling of relations within AutomationML Components the following rules shall apply:

- A relation shall be modelled by an InternalLink between Interfaces of two relation partners
Note: Multiple relations between two relation partners can be realized by adding InternalLinks
- If the relation has no direction two InternalLinks shall be defined in each direction.
- If the relation has a direction it is directed from “RefSideA” to “RefSideB” of the InternalLink.

3.14.1.1 Relations between Connectors and Models within an AutomationML Component

The connectors of an AutomationML Component represent virtual or real external interfaces of the real Automation component. Mostly the connectors have a relation to the data models inside the AutomationML Component. For instance, a “MechanicConnector”, like the flange on a movable piston rod, has a relation/influence on the kinematic model of the Component, because if it is moved by an external force, the piston rod might move alongside.

For the modelling of the relations between Connectors and Models within an AutomationML Component the following rules shall apply:

- The relation shall be modelled by an InternalLink between an Interface of the Connector and an Interface of the model.
- The linked connector and the linked model shall have a role requirement of a role class defined in the role class libraries “AutomationMLComponentBaseRCL” or “AutomationMLComponentStandardRCL” or any derived.

The following table shows some examples of possible relations between Connectors and Models and their interfaces:

Table 3: Examples for possible Connector to Model relations

Connector	Connector Interface	Model	Model Interface
ElectricConnector	ElectricInterface	BehaviourModel	VariableInterface
		GeometryModel	AttachmentInterface
		KinematicModel	AttachmentInterface
PneumaticConnector	PneumaticInterface	BehaviourModel	VariableInterface
		GeometryModel	AttachmentInterface
		KinematicModel	AttachmentInterface
LogicConnector	Signal Interface	BehaviourModel	VariableInterface
MechanicConnector	MechanicInterface	KinematicModel	AttachmentInterface
Skill Connector	Skill Interface	BehaviourModel	
...

3.14.1.2 Relations between Models within an AutomationML Component

The Models of an AutomationML Component represent data models for a specific domain or use case of Automation component. This section describes the modelling of the relation of different data models within an AutomationML Component.

For the modelling of relations between Models within AutomationML Component the following rules shall apply:

- The relation shall be modelled by an InternalLink between Interfaces of two models.
- The linked models shall have a role requirement of a role class defined in the role class libraries “AutomationMLComponentBaseRCL” or “AutomationMLComponentStandardRCL” or any derived.

An example for a relation between models can be found in the chapter about kinematics. There a relation between the variable interface of logics is connected to the JointInterface of a COLLADAKinematicJoint. This interdisciplinary connects logics with kinematics and provide an added value that can simplify the use of AutomationML Components e.g. in virtual commissioning software tools.

3.14.2 Composite Components: Relations between different AutomationML Components

Two or more AutomationML Components or Composite Components can be interconnected using connectors in different or the same domain, building Composite Components.

Figure 94 shows an example how to build a Composite Component out of single components. Multiple single AutomationML Components connected with Internal Links on their Connectors build a resulting AutomationML Composite Component with its own set of Connectors.

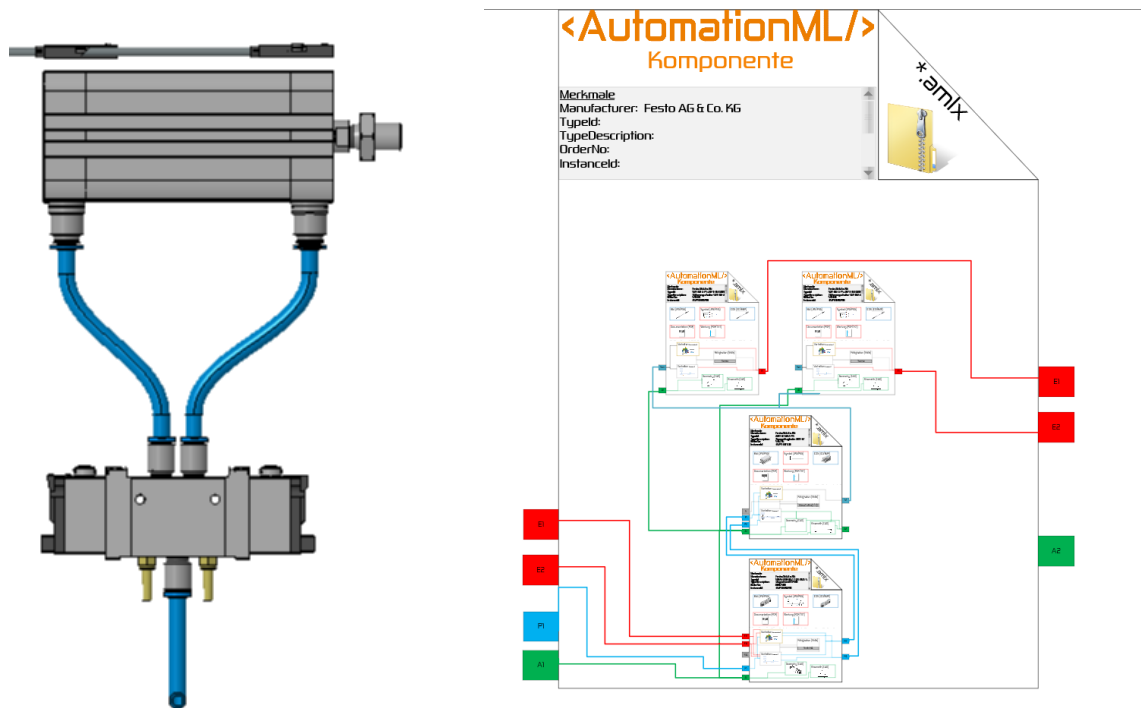


Figure 94: CAD Model of a composite automation component and a graphical visualization of the belonging AutomationML Component

For the interconnection of AutomationML Components or Composite Components via their connectors the following rules shall apply:

- Two or more AutomationML Components shall be connected with InternalLinks between the Interfaces of Connectors.

The Connectors shall have a role requirement of a role class defined in the role class libraries "AutomationMLComponentBaseRCL" or "AutomationMLComponentStandardRCL" or any derived.

4 Standard Libraries

4.1 Role Class Libraries

4.1.1 Overview new Role Class Libraries

Basement of the modelling AutomationML Components are the required role classes. Facing the addressed aspects of AutomationML Components a set of role classes are defined in this whitepaper.

Therefore, three AutomationML role class libraries are defined. The name of the first role class lib is "AutomationMLComponentBaseRCL". Within the role class library basic abstract role classes are defined that in general should not be used within an AutomationML Component description.

The second role class library is the "AutomationMLComponentStandardRCL" the role classes within the library cover the concrete model and information aspects of an AutomationML Component. They shall be used to within the definition AutomationML Components as InternalElements or SystemUnitClass.

The third role class library has the name "AutomationMLFMILogicRoleClassLib". This library covers some basic definitions to integrate FMU/FMI information into AutomationML structures. It is planned to standardize the library within the next maintenance cycle of the IEC representation of [WP-Part4:2018].

The role classes are derived from role classes defined from AutomationML basic roles defined in AutomationML Whitepaper – Architecture and general requirements and role classes defined in [WP-Part1:2018].

Figure 95 gives an overview about the AutomationML role class libraries "AutomationMLComponentBaseRCL" and "AutomationMLComponentStandardRCL" and the relation between their role classes.

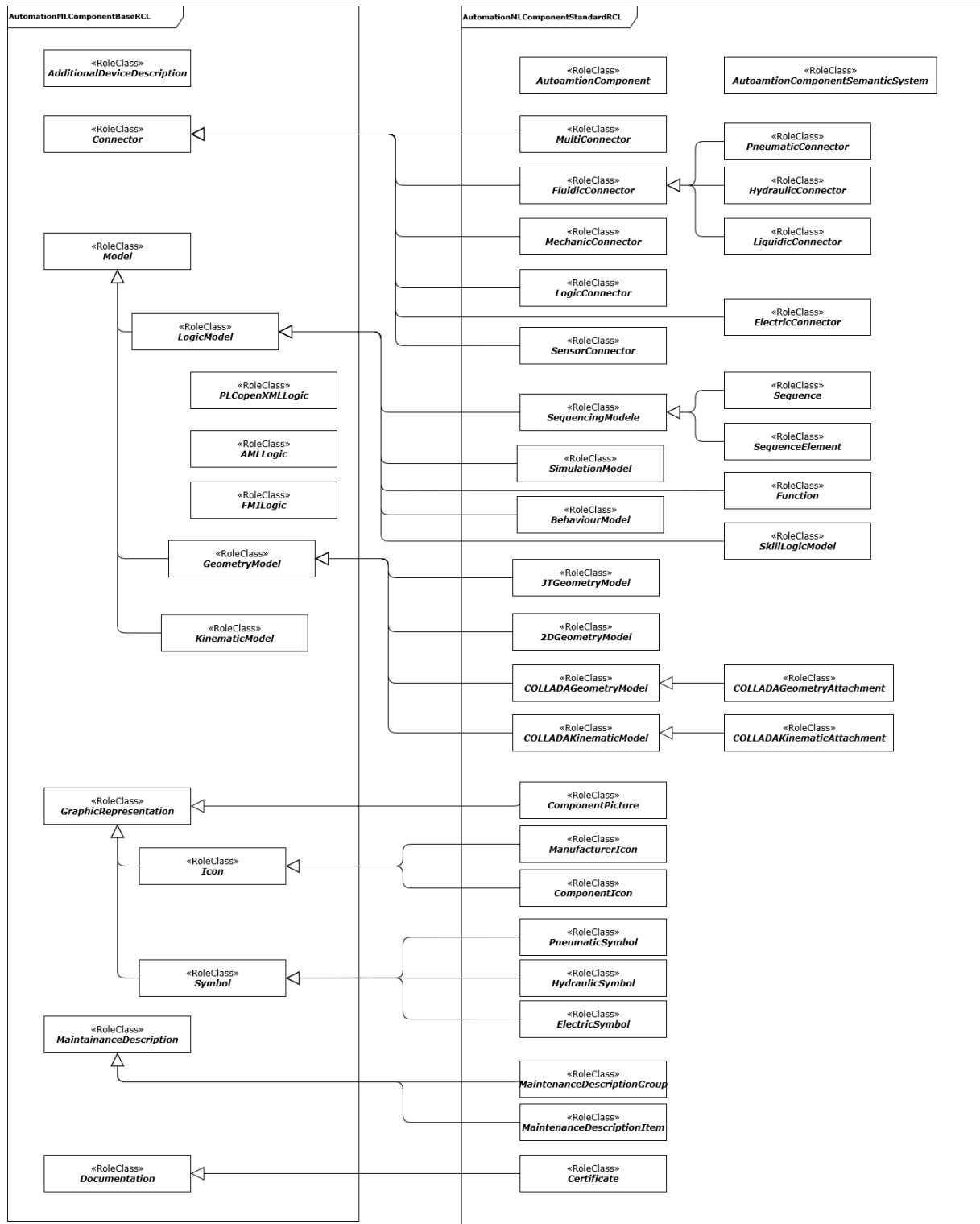


Figure 95: Inheritance structure of "AutomationMLComponentBaseRCL" and "AutomationMLComponentStandardRCL"

4.1.2 AutomationMLComponentBaseRCL

Figure 96 present the normative role class library “AutomationMLComponentBaseRCL” as object tree. Within this role class library abstract and base role classes are defined. This role classes are the parent classes for the standard role classes specified in chapter 4.1.3.

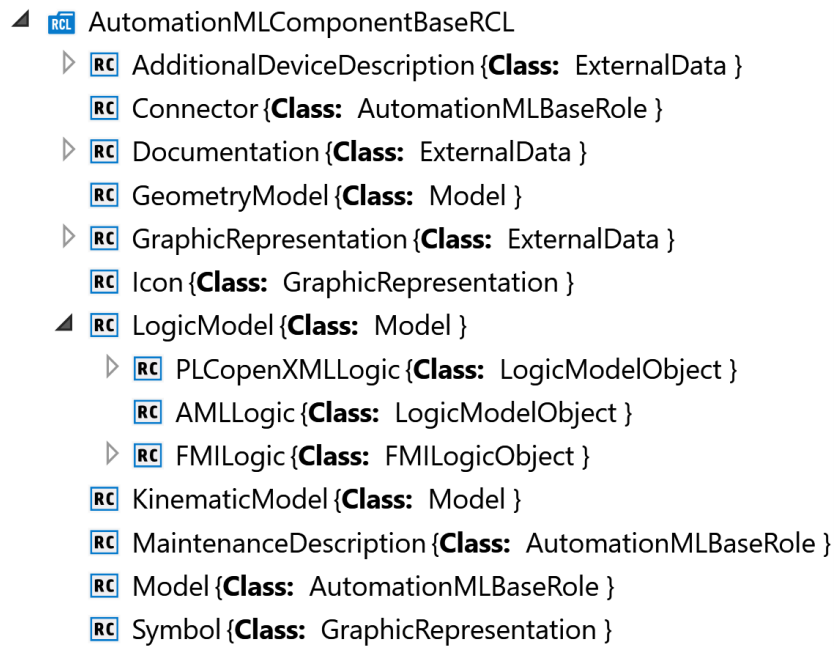


Figure 96: Overview “AutomationMLComponentBaseRCL”

4.1.2.1 RoleClass AdditionalDeviceDescription

The role class “AdditionalDeviceDescription” shall be used as specified in Table 4.

Table 4: RoleClass AdditionalDeviceDescription

Class name	AdditionalDeviceDescription	
Description	This is the base class for standard or user defined role classes referencing technology-based device descriptions.	
Parent Class	AutomationMLBPRRoleClassLib/ExternalData	
Path for element reference	AutomationMLComponentBaseRCL/AdditionalDeviceDescription	
Attributes	“SpecVersion” xs:string	The version of the Specification or Schema of the device description file.
	“DocLang” xs:string	Language of the referenced description file according to [RFC5646:2009]. The attribute is optional.
Interfaces	“DeviceDescriptionReference” (Path of Interface Type = “AutomationMLComponentBaseRCL/DeviceDescriptionReference”)	This interface is used to reference a to a technology related device description file according to the respective technology standard of an AutomationML Component. The use of the interface shall be mandatory.

4.1.2.2 RoleClass Connector

The role class “Connector” shall be used as specified Table 5.

Table 5: RoleClass Connector

Class name	Connector
Description	The role class “Connector” is an abstract basic role class and the base class for standard or user defined role classes describing connectors of components.
Parent class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole
Path for element reference	AutomationMLComponentBaseRCL/Connector
Attribute	None
InterfaceClass	None

4.1.2.3 RoleClass Documentation

The role class “Documentation” shall be used as specified in Table 6.

Table 6: RoleClass Documentation

Class name	Documentation	
Description	The role class “Documentation” shall be used to specify all AutomationML objects referencing documentation related information of a component.	
Parent class	AutomationMLBPRRoleClassLib/ExternalData	
Path for element reference	AutomationMLComponentBaseRCL/Documentation	
Attribute	None	
Interfaces	“ExternalDataReference” (Path of Interface Type = “AutomationMLBPRInterfaceClassLi b/ExternalDataReference”)	This interface is used to reference the documentation of a component. The use of the interface shall be mandatory and shall have the cardinality [1...n].

Note 1: All provisions of the [BPR-EDRef:2016] shall be used for role class “Documentation” and all derived classes used in the context of Automation components.

4.1.2.4 RoleClass GeometryModel

The role class “GeometryModel” shall be used as specified in Table 7.

Table 7: RoleClass GeometryModel

Class name	GeometryModel
Description	The role class “GeometryModel” is a basic abstract role class and the base class for standard or user defined role classes referencing component geometry models.
Parent Class	AutomationMLComponentBaseRCL/Model
Path for element reference	AutomationMLComponentBaseRCL/GeometryModel
Attributes	None
Interfaces	None

4.1.2.5 RoleClass GraphicRepresentation

The role class “GraphicRepresentation” shall be used as specified in Table 8.

Table 8: RoleClass GraphicRepresentation

Class name	GraphicRepresentation	
Description	The role class “GraphicRepresentation” is a basic role class and the base class for standard or user defined role classes referencing graphic representation of components.	
Parent Class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole	
Path for element reference	AutomationMLComponentBaseRCL/GraphicRepresentation	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (Path of Interface Type = “AutomationMLComponentBaseRCL/ GraphicRepresentationReference”)	This interface is used to reference a graphic of an AutomationML Component. The use of the interface shall mandatory and shall have the cardinality [1...n].

4.1.2.6 RoleClass Icon

The role class “Icon” shall be used as specified in Table 9.

Table 9: RoleClass Icon

Class name	Icon	
Description	The role class “Icon” is a basic role class and the base class for standard or user defined role classes referencing icons of components.	
Parent class	AutomationMLComponentBaseRCL/GraphicRepresentation	
Path for element reference	AutomationMLComponentBaseRCL/Icon	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.2.7 RoleClass LogicModel

The role class “LogicModel” shall be used as specified in Table 10.

Table 10: RoleClass LogicModel

Class name	LogicModel	
Description	The role class “LogicModel” is a basic abstract role class and the base class for standard or user defined role classes referencing component logic models.	
Parent class	AutomationMLComponentBaseRCL/Model	
Path for element reference	AutomationMLComponentBaseRCL/LogicModel	
Attributes	None	
Interfaces	None	

4.1.2.8 RoleClass PLCopenXMLLogic

The role class “PLCopenXMLLogic” shall be used as specified in Table 11.

Table 11: RoleClass PLCopenXMLLogic

Class name	PLCopenXMLLogic	
Description	The role class “PLCopenXMLLogic” shall be used for referencing a PLCopenXMLLogic model.	
Parent class	AutomationMLLogicRoleClassLib/LogicModelObject	
Path for element reference	AutomationMLComponentBaseRCL/LogicModel/PLCopenXMLLogic	
Attributes	None	
Interfaces	VariableInterface (Path of Interface Type = “AutomationMLPLCopenXMLInterfaceClassLib/VariableInterface”)	This interface shall be used to reference PLCopen XML variables of an PLCopen logic model that are used within an AutomationML Component. The use of the interface shall be optional and shall have the cardinality [0..n].
	LogicInterface (Path of Interface Type = “AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/LogicInterface”)	This interface shall be used to reference PLCopen XML logic model. The use of the interface shall be mandatory and shall have the cardinality [1].
	ReferenceSequence (inherited)	This interface shall NOT be used to reference PLCopen XML logic model. The interface has the cardinality [0].
	ReferenceBehaviour (inherited)	This interface shall NOT be used to reference PLCopen XML logic model. The interface has the cardinality [0]. The use of the interface is optional and has the cardinality [0..1].

4.1.2.9 RoleClass AMLLogic

The role class “AMLLogic” shall be used as specified in Table 12.

Table 12: RoleClass AMLLogic

Class name	AMLLogic	
Description	The role class “AMLLogic” shall be used for referencing an AMLLogic model.	
Parent class	AutomationMLLogicRoleClassLib/LogicModelObject	
Path for element reference	AutomationMLComponentBaseRCL/LogicModel/AMLLogic	
Attributes	None	
Interfaces	ReferenceSequence (inherited)	This interface shall be used to reference AML logic model. The use of the interface is optional and has the cardinality [0..1].
	ReferenceBehaviour (inherited)	This interface shall be used to reference AML logic model. The use of the interface is optional and has the cardinality [0..1].

Note: The use of one interface is mandatory and both interfaces have an xor relationship.

4.1.2.10 RoleClass FMILogic

The role class “FMILogic” shall be used as specified in Table 13.

Table 13: RoleClass FMILogic

Class name	FMILogic	
Description	<p>This RoleClass “FMILogic” describes an instance of a co-simulation Functional Mockup Unit (FMU) according to the FMI standard (see https://fmi-standard.org/), which is an open standard and is adopted by a variety of simulation tools.</p> <p>The type relation is modeled via a mandatory FMILogicReference, which is an AutomationML interface attached to the InternalElement referencing this FMISimulationModel-RoleClass.</p>	
Parent class	AutomationMLFMILogicRoleClassLib/FMILogicObject	
Path for element reference	AutomationMLComponentBaseRCL/LogicModel/FMILogic	
Attributes	“Name” xs:string	This is the instance name of the FMU.
	“Description” xs:string	This is the description of the FMU instance. It should describe unique features (e.g. regarding performance or precision) of this simulation model since multiple FMISimulationModels may be

		attached to an InternalElement representing an automation component.
	“FMIVersion” xs:string	The FMI version used by the referenced FMU. By today it must be one character string out of the set [“v1.0”, “v2.0”, “v3.0”]
Interfaces	FMIReference (Path of Interface Type = “AutomationMLFMIInterfaceClassLib/FMIReference”)	This interface is used to connect the logical behaviour of the FMI model to the component description. The use of the interface is mandatory and shall have the cardinality [1].
	FMIVariableInterface (Path of Interface Type = “AutomationMLFMIInterfaceClassLib/FMIVariableInterface”)	This interface is used to connect the variables for the specified FMI variable. The use of the interface shall be mandatory and shall have the cardinality [1...n].

4.1.2.11 RoleClass KinematicModel

The role class “KinematicModel” shall be used as specified in Table 14.

Table 14: RoleClass KinematicModel

Class name	KinematicModel
Description	The basic abstract role class “KinematicModel” shall be used as base class for standard or user defined role classes referencing component models.
Parent Class	AutomationMLComponentBaseRCL/Model
Path for element reference	AutomationMLComponentBaseRCL/KinematicModel
Attributes	None
Interfaces	None

4.1.2.12 RoleClass MaintenanceDescription

The role class “MaintenanceDescription” shall be used as specified Table 15.

Table 15: RoleClass MaintenanceDescription

Class name	MaintenanceDescription
Description	The role class “MaintenanceDescription” is an abstract basic role class and the base class for standard or user defined role classes defining maintenance description.
Parent class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole
Path for element reference	AutomationMLComponentBaseRCL/MaintenanceDescription
Attribute	None
InterfaceClass	None

4.1.2.13 RoleClass Model

The role class “Model” shall be used as specified in Table 16.

Table 16: RoleClass Model

Class name	Model
Description	The basic abstract role class “Function” shall be used as base class for standard or user defined role classes defining and referencing component models.
Parent class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole
Path for element reference	AutomationMLComponentBaseRCL/Model
Attribute	None
Interfaces	None

4.1.2.14 RoleClass Symbol

The role class “Symbol” shall be used as specified in Table 17.

Table 17: RoleClass Symbol

Class name	Symbol	
Description	The role class “Symbol” shall be used for referencing a symbol of the Automation component.	
Parent class	AutomationMLComponentBaseRCL/GraphicRepresentation	
Path for element reference	AutomationMLComponentStandardRCL/Symbol	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3 AutomationMLComponentStandardRCL

Figure 97 provides an overview about the “AutomationMLComponentStandardRCL” role class library as AutomationML object tree. The role classes shall be used to specify the single model and information aspects of AutomationML Components and Composite Components. All role classes of this role class library are derived for role classes of the “AutomationMLBaseRoleClassLib” or the “AutomationMLComponentBaseRCL”. The “AutomationMLBaseRoleClassLib” is defined in [WP-Part1:2018] and the “AutomationMLComponentBaseRCL”. Is defined in chapter 4.1.2 of this document.

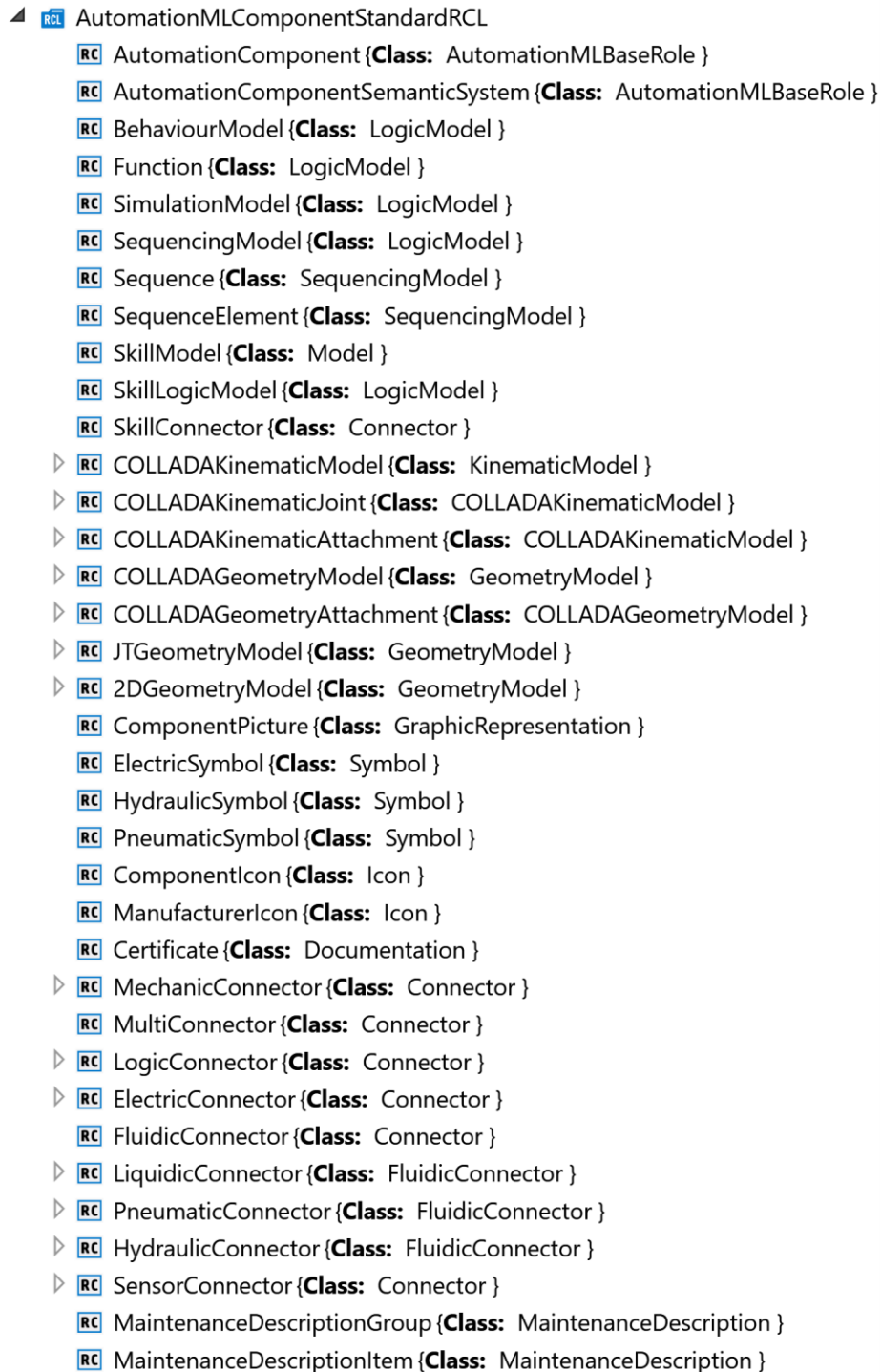


Figure 97: Overview AutomationMLComponentStandardRCL as AutomationML tree

4.1.3.1 RoleClass AutomationComponent

The role class “AutomationComponent” shall be used as specified in Table 18.

Table 18: RoleClass AutomationComponent

Role class	AutomationComponent	
Description	The role class “AutomationComponent” specifies the root element of an AutomationML Component. It defines a set of attributes to identify, classify and describe an industrial product which serves specific functions, i.e. for industrial process or factory automation.	
Parent class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole	
Path for element reference	AutomationMLComponentStandardRCL/AutomationComponent	
	“IdentificationData” complex	The attribute shall be used as complex attribute to specify the identification data of the automation component type or instance model. The attributes shall be used according to Table 19. The use of the attribute is optional.
	“GeneralTechnicalData” complex	The attribute shall be used as parent attribute for all technical classification attributes. The use of the attribute is optional.
	“CommercialData” complex	The attribute shall be used as parent attribute for all attributes defining commercial data. The attributes shall be used according to Table 20 to Table 22. The use of the attribute is optional.
	“ParameterData” complex	The attribute shall be used as parent attribute for parameter data (configurable and read-only) of the AutomationML Component. The use of the attribute is optional.
Interfaces	None	

The attribute “IdentificationData” shall be used as complex attribute specified in Table 19.

Table 19: Sub-Attributes “IdentificationData”

Attribute	AttributeDataType	Description
Manufacturer	xs:string	The attribute shall be used to define the manufacturer of the described AutomationML Component. The use of the attribute is optional.
ManufacturerURI	xs:string	The attribute shall be used to define the manufacturer URI of the described AutomationML Component. The use of the attribute is optional.
DeviceClass	xs:string	The attribute shall be used to define the device class or product family of the described AutomationML Component. The use of the attribute is optional.

Model	xs:string	The attribute shall be used to define the product or model name of the described AutomationML Component. The use of the attribute is optional.
ProductCode	xs:string	The attribute shall be used to define the unique product code of the described AutomationML Component. The use of the attribute is optional.
OrderCode	xs:string	The attribute shall be used to define the unique order code of the described AutomationML Component. The use of the attribute is optional.
HardwareRevision	xs:string	The attribute shall be used to define the hardware revision of the described AutomationML Component. The use of the attribute is optional.
SoftwareRevision	xs:string	The attribute shall be used to define the software or firmware revision of the described AutomationML Component. The use of the attribute is optional.
SerialNumber	xs:string	The attribute shall be used to define the serial number of the described AutomationML Component instance model. The use of the attribute is optional.
FabricationNumber	xs:string	The attribute shall be used to trace the date, time and circumstances of the fabrication of the described AutomationML Component instance model. The use of the attribute is optional.
ProductInstanceURI	xs:string	The ProductInstanceURI shall be used as defined in the [OPCUA-Part100:2020]

The attribute “CommercialData” shall be used as complex attribute as specified in Table 19.

Table 20: Sub-Attributes “CommercialData”

Attribute	AttributeDataType	Description
PackagingAndTransportation	complex	The sub attributes of the attribute shall be used to define the properties characterizing the packing and transportation (shipping) of a product. The use of the attribute is optional.
ProductDetails	complex	The sub attributes of the attribute shall be used to define the product details of the automation component. The sub attributes shall follow the definitions of Table 21. The use of the attribute is optional.
ProductOrderDetails	complex	The sub attributes of the attribute shall be

		used to define the product order details of the automation component. The sub attributes shall follow the definitions of Table 22. The use of the attribute is optional.
ProductPriceDetails	complex	The sub attributes of the attribute shall be used to define the product price details of the automation component. The sub attributes shall follow the definitions of Table 23. The use of the attribute is optional.
ManufacturerDetails	complex	The sub attributes of the attribute shall be used to define the product price details of the automation component. The sub attributes shall follow the definitions of Table 25. The use of the attribute is optional.

The attribute “ProductDetails” shall be used as complex attribute as specified in Table 21.

Table 21: Sub-Attributs “ProductDetails”

Attribute	AttributeDataType	Description
DescriptionShort	xs:string	Contains a short description / designation of the product (single line)
DescriptionLong	xs:string	Contains a technical description / designation of the product (multiple lines)
InternationalPID	xs:string	Contains the international Order number of the product according to the used standard
InternationalPIDType	xs:string	Contains the abbreviation of the international PID standard (e.g. gtin, EAN...)
ManufacturerPID	xs:string	Contains the order number of the original manufacturer
SpecialTreatmentClass	xs:string The attribute shall be defined as unordered list type according to [BPR-MLA:2016]	Additional product classification used for hazardous goods or substances, primary pharmaceutical products, radioactive measuring equipment, etc. The “type” attribute specifies the dangerous goods classification scheme. The value shall contain a special treatment class. Additional the RefSemantic Element shall be used to specify the system that defines the used treatment classes. The use of the attribute is optional.
Keyword	xs:string The attribute shall be defined as unordered list type according to [BPR-MLA:2016]	Keyword that supports product search in target systems.
Remarks	xs:string	Remark related to a business document.

The attribute “ProductOrderDetails” shall be used as complex attribute as specified in Table 22.

Table 22: Sub-Attributs “ProductOrderDetails”

Attribute	AttributeDataType	Description
OrderUnit	xs:string	Unit in which the product can be ordered; it is only possible to order multiples of the product unit.
ContentUnit	xs:string	Unit of the product related to the order unit
PriceQuantity	xs:string	If nothing is specified in this field the default value 1 is assumed, in other words the price refers to exactly one order unit. If specified, a multiple or a fraction of the order unit (element ORDER_UNIT) which indicates the quantity to which all the specified prices refer.
QuantityMin	xs:float	Minimum order quantity with respect to the order unit (ORDER_UNIT); if not specified, the minimum order quantity is 1.
QuantityInterval	xs:float	Maximum order quantity with respect to the order unit (ORDER_UNIT); if not specified, the order quantity is not limited.
QuantityMax	xs:float	Number indicating the quantity steps in which the articles can be ordered. The first step always corresponds to the minimum order quantity specified. The unit of the quantity interval is the same as the order unit.
PackingUnits	xs:string	Information on the dependency of the packing unit from the order unit. Example: Printing paper á 500 sheets has the order unit pack; ordering 5 packs results in a new packing unit, carton; ordering 50 packs or 10 cartons results in another packing unit, covering box; ordering 500 packs or 100 cartons results in the biggest packing unit here, palette.

Note: The use of the units shall follow the [BPR-Units:2018].

The attribute “ProductPriceDetails” shall be used as complex attribute as specified in Table 23.

Table 23: Sub-Attributs “ProductPriceDetails”

Attribute	AttributeDataType	Description
ValidStartDate	xs:date	The attribute shall be used to define the valid start date of the product price of the automation component. The use of the attribute is optional.
ValidEndDate	xs:date	The attribute shall be used to define the valid end date of the product price of the automation component. The use of the attribute is optional.
ProductPrice	complex	The sub attributes of the attribute shall be used to define the product price of the automation component. The sub attributes shall follow the definitions of Table 24. The use of the attribute is optional.

The attribute “ProductPrice” shall be used as complex attribute as specified in Table 24.

Table 24: Sub-Attributs “ProductPrice”

Attribute	AttributeDataType	Description
PriceAmount	xs:string	The attribute shall be used to define the price amount of the automation component. The use of the attribute is optional.
PriceCurrency	xs:string	The attribute shall be used to define the price currency of the product price of the automation component. The use of the attribute is optional.
Tax	xs:string	The attribute shall be used to define tax of the product price of the automation component. The use of the attribute is optional.
PriceFactor	xs:string	The attribute shall be used to define the price factor of the product price of the automation component. The use of the attribute is optional.
LowerBound	xs:string	The attribute shall be used to define the lower bound of the product price of the automation component. The use of the attribute is optional.
Territory	xs:string The attribute shall be defined as unordered list type according to [BPR-MLA:2016]	The attribute shall be used to define the lower bound of the product price of the automation component. The use of the attribute is optional.

The attribute “ManufacturerDetails” shall be used as complex attribute as specified in Table 25.

Table 25: Sub-Attributs “ManufacturerDetails”

Attribute	AttributeDataType	Description
Name	xs:string	The attribute shall be used to define the name of the manufacturer of the automation component. The use of the attribute is optional.
Address1	xs:string	The attribute shall be used to define the first address line of the manufacturer of the automation component. The use of the attribute is optional.
Address2	xs:string	The attribute shall be used to define the second address line of the manufacturer of the automation component. The use of the attribute is optional.
ZipCode	xs:string	The attribute shall be used to define the zip code of the manufacturer of the automation component. The use of the attribute is optional.
City	xs:string	The attribute shall be used to define the city of the manufacturer of the automation component. The use of the attribute is optional.
Country	xs:string	The attribute shall be used to define the country of

		the manufacturer of the automation component. The use of the attribute is optional.
ContactMail	xs:string	The attribute shall be used to define the contact mail of the manufacturer of the automation component. The use of the attribute is optional.
ContactPhone	xs:string	The attribute shall be used to define the contact phone number of the manufacturer of the automation component. The use of the attribute is optional.
Website	xs:string	The attribute shall be used to define the website of the manufacturer of the automation component. The use of the attribute is optional.

4.1.3.2 RoleClass AutomationComponentSemanticSystem

The role class “AutomationComponent” shall be used as specified in Table 18.

Table 26: RoleClass AutomationComponentSemanticSystem

Class name	AutomationComponentSemanticSystem	
Description	The role class “AutomationComponentSemanticSystem” shall be used to define the semantic systems that are supported by the AutomationML Component.	
Parent class	AutomationMLBaseRoleClassLib/ AutomationMLBaseRole	
Path for element reference	AutomationMLComponentStandardRCL/ AutomationComponentSemanticSystem	
Attributes	ClassificationSystem complex	The attribute “ClassificationSystem” shall be used to announce an own semantic system or standardized system that is used for the description of an AutomationML Component. It shall follow the provisions of Table 27. The attribute has the cardinality 0..n.
	IEC 61987 complex	The attribute “IEC 61987” shall be used to announce the IEC 61987 as semantic system for an AutomationML Component. It shall follow the provisions of Table 28. The attribute is optional.
	IEC 61360-4 complex	The attribute “IEC 61360-4” shall be used to announce the 61360-4 as semantic system for an AutomationML Component. It shall follow the provisions of Table 28. The attribute is optional.
	IEC 62683 complex	The attribute “IEC 62683” shall be used to announce the IEC 62683 as semantic system for an AutomationML Component. It shall follow the provisions of Table 28. The attribute is optional.

	eClass complex	The attribute “eClass” shall be used to announce an eClass as semantic system for an AutomationML Component. It shall follow the provisions of Table 28. The attribute is optional.
Interfaces	None	

The attribute “ClassificationSystem” shall be used as complex attribute specified in Table 27.

Table 27: Sub-attributes of the “ClassificationSystem” attribute

Attribute	AttributeDataType	Description
Name	xs:string	The attribute “Name” shall be used to specify the used classification system that is used to define attributes of the component in detail. The attribute is mandatory.
Version	xs:string	The attribute “Version” shall be used to specify particular version of the classification system that is used. The attribute is mandatory.
RefSemanticPrefix	xs:string	The attribute “RefSemanticPrefix” shall be used to specify the prefix used in the RefSemantic to define the semantic systems. The attribute is mandatory, and it shall follow the provisions of chapter 7.
URL	xs:string	The attribute “URL” shall be used to specify the URL where the semantic system can be found. The attribute is optional.

The values for the sub-attributes of the attributes “IEC 61987”, “IEC 61360-4”, “IEC 62683” and “eClass” shall be follow the specification in Table 28.

Table 28: Attribute values “SemanticSystems”

Sub-attribute	Values for attribute			
	IEC 61987	IEC 61360-4	IEC 62683	eClass
Version	2.0014.0016	2.0014.0016	2.0014.0016	11.0
RefSemanticPrefix	IRDI-PATH:0112/2///62683#	IRDI-PATH:0112/2///61360_4#	IRDI-PATH:0112/2///62683#	IRDI-PATH:0173
URL	https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/TreeFrameset?OpenFrameSet&ongletactif=1	https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet&ongletactif=1	https://cdd.iec.ch/cdd/iec62683/iec62683.nsf/TreeFrameset?OpenFrameSet&ongletactif=1	https://www.eclasccontent.com

4.1.3.3 RoleClass BehaviourModel

The role class “BehaviourModel” shall be used as specified in Table 29.

Table 29: RoleClass BehaviourModel

Class name	BehaviourModel
Description	The role class “BehaviourModel” shall be used in order to specify a logic model integration as behaviour model of AutomationML Component.
Parent class	AutomationMLComponentBaseRCL/LogicModel
Path for element reference	AutomationMLComponentStandardRCL/BehaviourModel
Attributes	None
Interfaces	None

4.1.3.4 RoleClass Function

The role class “Function” shall be used as specified in Table 30.

Table 30: RoleClass Function

Class name	Function
Description	The role class “Function” shall be used in order to specify a logic model integration as function of AutomationML Component.
Parent class	AutomationMLComponentBaseRCL/LogicModel
Path for element reference	AutomationMLComponentStandardRCL/Function
Attributes	None
Interfaces	None

4.1.3.5 RoleClass SimulationModel

The role class “SimulationModel” shall be used as specified in Table 31.

Table 31: RoleClass SimulationModel

Class name	SimulationModel
Description	The role class “SimulationModel” shall be used in order to specify a logic model integration as simulation model of AutomationML Component.
Parent class	AutomationMLComponentBaseRCL/LogicModel
Path for element reference	AutomationMLComponentStandardRCL/SimulationModel
Attributes	None
Interfaces	None

4.1.3.6 RoleClass SkillLogicModel

The role class “SkillLogicModel” shall be used as specified in Table 32.

Table 32: RoleClass SkillLogicModel

Class name	SkillLogicModel
Description	The role class “SkillLogicModel” shall be used in order to specify a logic model integration as behaviour model of AutomationML Component.
Parent class	AutomationMLComponentBaseRCL/LogicModel
Path for element reference	AutomationMLComponentStandardRCL/SkillLogicModel
Attribute	None
Interfaces	None

Note: The final definition of AutomationML Component skill models will be part of Version 2 of this document.

4.1.3.7 RoleClass SequencingModel

The role class “SequencingModel” shall be used as specified in Table 33

Table 33: RoleClass SequencingModel

Class name	SequencingModel
Description	The role class “SequencingModel” shall be used in order to specify all the interface related information of an sequence model of an AutomationML Component or Composite Component.
Parent class	AutomationMLComponentBaseRCL/LogicModel
Path for element reference	AutomationMLComponentStandardRCL/SequencingModel
Attributes	None
Interfaces	None

4.1.3.8 RoleClass Sequence

The role class “Sequence” shall be used as specified in Table 34.

Table 34: RoleClass Sequence

Class name	Sequence
Description	The role class “Sequence” shall be used in order to specify all the interface related information of a PLCopen XML sequence model.
Parent class	AutomationMLComponentStandardRCL/SequencingModel
Path for element reference	AutomationMLComponentStandardRCL/Sequence
Attribute	None
Interfaces	None

4.1.3.9 RoleClass SequenceElement

The role class “SequenceElement” shall be used as specified in Table 35.

Table 35: RoleClass SequenceElement

Class name	SequenceElement
Description	The role class “SequenceElement” shall be used in order to specify all the interface related information of a sequence element.
Parent class	AutomationMLComponentStandardRCL/SequencingModel
Path for element reference	AutomationMLComponentStandardRCL/SequenceElement
Attribute	None
Interfaces	None

4.1.3.10 RoleClass COLLADAKinematicModel

The role class “COLLADAKinematicModel” shall be used as specified in Table 36.

Table 36: RoleClass COLLADAKinematicModel

Class name	COLLADAKinematicModel	
Description	The role class “COLLADAKinematicModel” shall be used in order to specify all the interface related information of a COLLADA kinematic model of an AutomationML Component or Composite Component.	
Parent class	AutomationMLComponentBaseRCL/KinematicModel	
Path for element reference	AutomationMLComponentStandardRCL/COLLADAKinematicModel	
Attribute	None	
Interfaces	“COLLADAInterface” (RefBaseClassPath= “AutomationMLInterfaceClassLib/Auto mationMLBaseInterface/ExternalData Connector/COLLADAInterface”)	This interface is used to connect the COLLADA geometry model to the AutomationML Component or Composite Component. The “refURI” attribute of the interface shall reference to the origin of COLLADA Model.

4.1.3.11 RoleClass COLLADAKinematicJoint

The role class “COLLADAKinematicJoint” shall be used as specified in Table 37.

Table 37: RoleClass COLLADAKinematicJoint

Class name	COLLADAKinematicJoint	
Description	The role class “COLLADAKinematicJoint” shall be used in order to specify all joint information of a COLLADA kinematic model of the AutomationML Component.	
Parent class	AutomationMLComponentStandardRCL/COLLADAKinematicModel	
Path for element reference	AutomationMLComponentStandardRCL/COLLADAKinematicJoint	
Attribute	None	
Interfaces	“COLLADAInterface” (inherited)	Shall be used as described for parent class.
	“JointInterface” (RefBaseClassPath= “AutomationMLInterfaceClassLib/AutomationMLBaseInterface/AttachmentInterface”)	The interface shall be used to attach other interfaces that have a relation to the ColladaKinematicJoint.

4.1.3.12 RoleClass COLLADAKinematicAttachment

The role class “COLLADAKinematicAttachment” shall be used as specified in Table 37.

Table 38: RoleClass COLLADAKinematicAttachment

Class name	COLLADAKinematicAttachement	
Description	The role class “COLLADAKinematicAttachment” shall be used in order to specify all attachment information of a COLLADA kinematic model of the AutomationML Component.	
Parent class	AutomationMLComponentStandardRCL/COLLADAKinematicModel	
Path for element reference	AutomationMLComponentStandardRCL/COLLADAKinematicAttachmen	
Attribute	None	
Interfaces	“COLLADAInterface” (inherited)	Shall be used as described for parent class.
	“AttachmentInterface” (RefBaseClassPath= “AutomationMLInterfaceClassLib/AutomationMLBaseInterface/AttachmentInterface”)	The interface shall be used to attach coordinate systems of different COLLADA models.

4.1.3.13 RoleClass COLLADAGeometryModel

The role class “COLLADAGeometryModel” shall be used as specified in Table 39.

Table 39: RoleClass COLLADAGeometryModel

Class name	COLLADAGeometryModel	
Description	The role class “COLLADAGeometryModel” shall be used in order to specify all the interface related information of a COLLADA geometry model.	
Parent class	AutomationMLComponentBaseRCL/GeometryModel	
Path for element reference	AutomationMLComponentStandardRCL/COLLADAGeometryModel	
Attribute	None	
Interfaces	“COLLADAInterface” (RefBaseClassPath= “AutomationMLInterfaceClassLib/Auto mationMLBaseInterface/ExternalData Connector/COLLADAInterface”)	This interface is used to connect the COLLADA geometry model to the AutomationML Component or Composite Component. The “refURI” attribute of the interface shall reference to the origin of COLLADA Model.

4.1.3.14 RoleClass COLLADAGeometryAttachment

The role class “COLLADAGeometryAttachment” shall be used as specified in Table 39.

Table 40: RoleClass COLLADAGeometryAttachment

Class name	COLLADAGeometryAttachment	
Description	The role class “COLLADAGeometryAttachment” shall be used in order to specify all attachment information of a COLLADA geometry model of the AutomationML Component.	
Parent class	AutomationMLComponentStandardRCL/COLLADAGeometryModel	
Path for element reference	AutomationMLComponentStandardRCL/COLLADAGeometryAttachment	
Attribute	None	
Interfaces	“COLLADAInterface” (inherited)	Shall be used as described for parent class.
	“AttachmentInterface” (RefBaseClassPath= “ AutomationMLInterfaceClassLib/Auto mationMLBaseInterface/AttachmentIn terface”)	The interface shall be used to attach coordinate systems of different COLLADA models.

4.1.3.15 RoleClass JTGeometryModel

The role class “JTGeometryModel” shall be used as specified in Table 41.

Table 41: RoleClass JTGeometryModel

Class name	JTGeometryModel	
Description	The role class “JTGeometryModel” shall be used in order to specify all the interface related information of a JT geometry model.	
Parent class	AutomationMLComponentBaseRCL/GeometryModel	
Path for element reference	AutomationMLComponentStandardRCL/JTGeometryModel	
Attribute	None	
Interfaces	“ExternalDataConnector” (RefBaseClassPath= "AutomationML ComponentBaseICL/JTReference")	This interface is used to connect the JT geometry model to the AutomationML Component or Composite Component. The “refURI” attribute of the interface shall reference to the origin of JT Model.

4.1.3.16 RoleClass 2DGeometryModel

The role class “2DGeometryModel” shall be used as specified in Table 42.

Table 42: RoleClass 2DGeometryModel

Class name	2DGeometryModel	
Description	The role class “2DGeometryModel” shall be used in order to specify all the interface related information of a 2D geometry model.	
Parent class	AutomationMLComponentBaseRCL/GeometryModel	
Path for element reference	AutomationMLComponentStandardRCL/2DGeometryModel	
Attribute	None	
Interfaces	“2DReference” (RefBaseClassPath= "AutomationML ComponentBaseICL/ 2DReference")	This interface is used to connect the 2D model to the component. The “refURI” attribute of the interface shall reference to the origin of co- ordinates within the model if this exists within the referenced 2D model.

4.1.3.17 RoleClass ComponentPicture

The role class “ComponentPicture” shall be used as specified in Table 43.

Table 43: RoleClass ComponentPicture

Class name	ComponentPicture	
Description	The role class “ComponentPicture” shall be used for referencing a picture of the Automation component.	
Parent class	AutomationMLComponentBaseRCL/GraphicRepresentation	
Path for element reference	AutomationMLComponentStandardRCL/ComponentPicture	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.18 RoleClass ElectricSymbol

The role class “ElectricSymbol” shall be used as specified in Table 44

Table 44: RoleClass ElectricSymbol

Class name	ElectricSymbol	
Description	The role class “ElectricSymbol” shall be used for referencing a electric symbol of the Automation component.	
Parent class	AutomationMLComponentStandardRCL/Symbol	
Path for element reference	AutomationMLComponentStandardRCL/ElectricSymbol	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.19 RoleClass HydraulicSymbol

The role class “HydraulicSymbol” shall be used as specified in Table 45

Table 45: RoleClass HydraulicSymbol

Class name	HydraulicSymbol	
Description	The role class “HydraulicSymbol” shall be used for referencing a hydraulic symbol of the automation component.	
Parent class	AutomationMLComponentStandardRCL/Symbol	
Path for element reference	AutomationMLComponentStandardRCL/HydraulicSymbol	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.20 RoleClass PneumaticSymbol

The role class “PneumaticSymbol” shall be used as specified in Table 46

Table 46: RoleClass PneumaticSymbol

Class name	PneumaticSymbol	
Description	The role class “PneumaticSymbol” shall be used for referencing a pneumatic symbol of the automation component.	
Parent class	AutomationMLComponentStandardRCL/Symbol	
Path for element reference	AutomationMLComponentStandardRCL/PneumaticSymbol	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.21 RoleClass ManufactureIcon

The role class “ManufactureIcon” shall be used as specified in Table 47.

Table 47: RoleClass ManufactureIcon

Class name	ManufactureIcon	
Description	The role class “ComponentPicture” shall be used for referencing a picture of the Automation component.	
Parent class	AutomationMLComponentBaseRCL/Icon	
Path for element reference	AutomationMLComponentStandardRCL/ManufactureIcon	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.22 RoleClass ComponentIcon

The role class “ComponentIcon” shall be used as specified in Table 48.

Table 48: RoleClass ComponentIcon

Class name	ComponentIcon	
Description	The role class “ComponentIcon” shall be used for referencing a icon of the automation component.	
Parent class	AutomationMLComponentBaseRCL/Icon	
Path for element reference	AutomationMLComponentStandardRCL/ComponentIcon	
Attribute	None	
Interfaces	“GraphicRepresentationReference” (inherited)	Shall be used as described for parent class.

4.1.3.23 RoleClass SkillModel

The role class “SkillModel” shall be used as specified in Table 49.

Table 49: RoleClass SkillModel

Class name	SkillModel
Description	The role class “SkillModel” shall be used to define or reference skill models of an automation Component.
Parent class	AutomationMLComponentBaseRCL/Model
Path for element reference	AutomationMLComponentStandardRCL/SkillModel
Attribute	None
Interfaces	None

Note: The final definition of AutomationML Component skill models will be part of Version 2 of this document.

4.1.3.24 RoleClass Certificate

The role class “Certificate” shall be used as specified in Table 48.

Table 50: RoleClass Certificate

Class name	Certificate		
Description	The role class “Certificate” shall be used for referencing a certification document of the automation component.		
Parent class	AutomationMLComponentBaseRCL/Documentation		
Path for element reference	AutomationMLComponentStandardRCL/Certificate		
Attribute	None		
Interfaces	<table border="1"> <tr> <td>“ExternalDataReference” (inherited)</td><td>Shall be used as described for parent class.</td></tr> </table>	“ExternalDataReference” (inherited)	Shall be used as described for parent class.
“ExternalDataReference” (inherited)	Shall be used as described for parent class.		

4.1.3.25 RoleClass MechanicConnector

The role class “MechanicConnector” shall be used as specified in Table 51

Table 51: RoleClass MechanicConnector

Class name	MechanicConnector		
Description	The role class “MechanicConnector” shall be used to define the representation of a mechanical fastening interface of an automation component within its AutomationML Component representation.		
Parent class	AutomationMLComponentBaseRCL/Connector		
Path for element reference	AutomationMLComponentStandardRCL/MechanicConnector		
Attribute	None		
Interfaces	<table border="1"> <tr> <td>“MechanicInterface” (RefBaseClassPath= “AutomationML ComponentBaseICL/MechanicInterfac e”)</td><td>This interface is used to connect the mechanic connectors of components. The use of the interface shall be 1..n.</td></tr> </table>	“MechanicInterface” (RefBaseClassPath= “AutomationML ComponentBaseICL/MechanicInterfac e”)	This interface is used to connect the mechanic connectors of components. The use of the interface shall be 1..n.
“MechanicInterface” (RefBaseClassPath= “AutomationML ComponentBaseICL/MechanicInterfac e”)	This interface is used to connect the mechanic connectors of components. The use of the interface shall be 1..n.		

4.1.3.26 RoleClass LogicConnector

The role class “LogicConnector” shall be used as specified in Table 52.

Table 52: RoleClass LogicConnector

Class name	LogicConnector	
Description	The role class “LogicConnector” shall be used in order to specify all the related information of an logic connector of an automation component within its AutomationML Component representation.	
Parent class	AutomationMLComponentBaseRCL/Connector	
Path for element reference	AutomationMLComponentStandardRCL/LogicConnector	
Attribute	None	
Interfaces	“SignalInterface” (RefBaseClassPath=AutomationMLInterfaceClassLib/AutomationMLBaseInterface/Communication/SignalInterface) “	This interface is used to connect the logic connectors of components. The use of the interface shall be 1..n.

4.1.3.27 RoleClass ElectricConnector

The role class “ElectricConnector” shall be used as specified in Table 53.

Table 53: RoleClass ElectricConnector

Class name	ElectricConnector	
Description	The role class “ElectricConnector” shall be used in order to specify all the related information of an electric connector of an automation component within its AutomationML Component representation.	
Parent class	AutomationMLComponentBaseRCL/Connector	
Path for element reference	AutomationMLComponentStandardRCL/ElectricConnector	
Attribute	None	
Interfaces	“ElectricInterface” (RefBaseClassPath=AutomationMLComponentBaseICL/ElectricInterface) “	This interface is used to connect the electric connectors of components. The use of the interface shall be 1..n.

4.1.3.28 RoleClass FluidicConnector

The role class “FluidicConnector” shall be used as specified in Table 54.

Table 54: RoleClass FluidicConnector

Class name	FluidicConnector	
Description	The role class “FluidicConnector” is an abstract role class and the base class for standard or user defined role classes referencing fluidic connectors.	
Parent class	AutomationMLComponentBaseRCL/Connector	
Path for element reference	AutomationMLComponentStandardRCL/FluidicConnector	
Attribute	None	

Interfaces	None
-------------------	------

4.1.3.29 RoleClass LiquidicConnector

The role class “LiquidicConnector” shall be used as specified in Table 55.

Table 55: RoleClass LiquidicConnector

Class name	LiquidicConnector	
Description	The role class “LiquidicConnector” shall be used in order to specify all the related information of a liquidic connector of an automation component within its AutomationML Component representation.	
Parent class	AutomationMLComponentBaseICL/LiquidicInterface	
Path for element reference	AutomationMLComponentStandardRCL/LiquidicConnector	
Attribute	None	
Interfaces	“LiquidicInterface” (RefBaseClassPath=AutomationMLComponentBaseICL/LiquidicInterface)”	This interface is used to connect the liquidic connectors of components. The use of the interface shall be 1..n.

4.1.3.30 RoleClass PneumaticConnector

The role class “PneumaticConnector” shall be used as specified in Table 56.

Table 56: RoleClass PneumaticConnector

Class name	PneumaticConnector	
Description	The role class “PneumaticConnector” shall be used in order to specify all the related information of a pneumatic connector of an automation component within its AutomationML Component representation.	
Parent class	AutomationMLComponentStandardRCL/FluidicConnector	
Path for element reference	AutomationMLComponentStandardRCL/PneumaticConnector	
Attribute	None	
Interfaces	“PneumaticConnector” (RefBaseClassPath=AutomationMLComponentBaseICL/PneumaticInterface)”	This interface is used to connect the pneumatic connectors of components. The use of the interface shall be 1..n.

4.1.3.31 RoleClass HydraulicConnector

The role class “HydraulicConnector” shall be used as specified in Table 57.

Table 57: RoleClass HydraulicConnector

Class name	HydraulicConnector	
Description	The role class “HydraulicConnector” shall be used in order to specify all the related information of a hydraulic connector of an automation component within its AutomationML Component representation.	
Parent class	AutomationMLComponentStandardRCL/FluidicConnector	
Path for element reference	AutomationMLComponentStandardRCL/HydraulicConnector	
Attribute	None	
Interfaces	“HydraulicInterface” (RefBaseClassPath=AutomationMLComponentBaseICL/HydraulicInterface)”	This interface is used to connect the hydraulic connectors of components. The use of the interface shall be 1..n.

4.1.3.32 RoleClass SensorConnector

The role class “SensorConnector” shall be used as specified in Table 58.

Table 58: RoleClass SensorConnector

Class name	SensorConnector
Description	SensorConnector is the process interface to physically sense the properties of interest (i.e. a mechanical movement or object presence within a spatial detection area or mediums states like temperature or pressure). It is used for both mechanical construction and simulation environments.
Parent class	AutomationMLComponentBaseRCL/Connector
Path for element reference	AutomationMLComponentStandardRCL/SensorConnector
Attributes	Will be defined in the next version of this document.
Interfaces	Will be defined in the next version of this document.

4.1.3.33 RoleClass SkillConnector

The role class “SkillConnector” shall be used as specified in Table 59.

Table 59: RoleClass SkillConnector

Class name	SkillConnector
Description	SkillConnector is used as an engineering interface for skill-based systems engineering. SkillConnector provides interfaces necessary to connect the components logically in an automation system and also to connect with products and process in a manufacturing plant
Parent class	AutomationMLComponentStandardRCL/SkillConnector
Path for element reference	AutomationMLComponentStandardRCL/SkillConnector
Attribute	None
Interfaces	None

4.1.3.34 RoleClass MaintenanceDescriptionGroup

The role class “MaintenanceDescriptionGroup” shall be used as specified in Table 56.

Table 60: RoleClass MaintenanceDescriptionGroup

Class name	MaintenanceDescriptionGroup
Description	The role class “MaintenanceDescriptionGroup” shall be used to define a assembly for different maintenance tasks.
Parent class	AutomationMLComponentBaseRCL/MaintenanceDescription
Path for element reference	AutomationMLComponentStandardRCL/MaintenanceDescriptionGroup
Attribute	<div> <div> “TopicName” xs:string </div> <div> Each maintenance measure shall be assigned to a major assembly/heading. The attribute shall follow the provisions of [BPR-MLA:2016] for multilingual expressions. </div> </div>
Interfaces	None

4.1.3.35 RoleClass MaintenanceDescriptionItem

The role class “MaintenanceDescriptionItem” shall be used as specified in Table 56.

Table 61: RoleClass MaintenanceDescriptionItem

Class name	MaintenanceDescriptionItem	
Description	The role class “MaintenanceDescriptionItem” shall be used to define a single maintenance task.	
Parent class	AutomationMLComponentBaseRCL/MaintenanceDescription	
Path for element reference	AutomationMLComponentStandardRCL/MaintenanceDescriptionItem	
Attribute	“Index” xs:string	Every maintenance item shall be given a procedure number.
	“SubTopic” xs:string	If the maintenance item refers to sub topic within the Topic of the group, this shall be listed The attribute shall follow the provisions of [BPR-MLA:2016] for multilingual expressions.
	“WorkDescription” xs:string	Description of the activity to be performed. The attribute shall follow the provisions of [BPR-MLA:2016] for multilingual expressions.
	“Cycle” xs:duration	Every maintenance item is to be assigned to one of the maintenance cycles listed Table 62
	“PlannedTimePerWorker” xs:duration	Shall be used to define the plan value for execution of the maintenance item.
	“ActivityKey” xs:string	Every maintenance measure is to be assigned to one of the occupation codes listed in Table 63
	“ExecutionKey” xs:string	Every maintenance measure is to be assigned to one of the occupation codes listed in Table 64
	“FunctionKey” xs:string	Every maintenance measure is to be assigned to one of the function keys listed in Table 65
	“PersonnelKey” xs:string	Every maintenance measure is to be assigned to one of the personnel keys listed in Table 66
	“LastExecution” xs:string	Shall be used to describe the last execution of the maintenance item.
Interfaces	None	

Table 62: Values for Attribute “Cycle”

Value	Significance	Value	Significance
-------	--------------	-------	--------------

1S	Every shift	18M	Every 18 months
1D	Daily	2Y	Every 2 years
3T	Every 3 workdays	3Y	Every 3 years
5D	Every 5 workdays	4Y	Every 4 years
1W	weekly	5Y	Every 5 years
2W	Every 2 weeks	6Y	Every 6 years
3W	Every 3 weeks	7Y	Every 7 years
1M	Monthly	8Y	Every 8 years
2M	Every 2 months	9Y	Every 9 years
3M	Every 3 months	10Y	Every 10 years
4M	Every 4 months	12Y	Every 12 years
6M	Every 6 months	15Y	Every 15 years
9M	Every 9 months	20Y	Every 20 years
1Y	Annually	25Y	Every 25 years

Table 63: Values for Attribute "ActivityKey"

Value	Significance
inspection	Measures for determining or assessing the actual status, such as the inspection for wear, corrosion, leaks, connections or periodic measurement and evaluation.
maintenance	Measures for maintaining the required state, such as cleaning, lubricating, adjusting or calibrating.
overhaul	Measures for restoring the required state, such as the disassembly and replacement of components and assemblies.

Table 64: Values for Attribute "ExecutionKey"

Values	Significance
production	PM measures that can be taken during production.
standstill	PM measures that can only be performed during a system downtime (outside production hours).

Table 65: Values for Attribute "FunctionKey"

Values	Significance
availability	Maintenance measures that are mainly required for ensuring the technical availability of plants, devices and components.
environmental	Maintenance measures that are mainly required for ensuring environmental protection for environmentally relevant plants, devices and components.
safety	Maintenance measures that are mainly required for occupational safety and for preventing accidents.

Table 66: Values for Attribute "PersonnelKey"

Values	Significance
--------	--------------

mechanic	Assignment of the measures that can primarily be performed by trained industrial mechanics.
electrician	Assignment of the measures which can primarily be performed by trained industrial electricians.
operator	Assignment of the measures which can primarily be performed by technically skilled and semi-skilled production personnel with corresponding qualifications.
qualified person	Assignment of the measures which can be carried out in the sense of the operating safety directive by persons who, through their vocational training, their professional experience and their professional activity, have the specialist knowledge required to test the operating equipment.

4.1.4 AutomationMLFMILogicRoleClassLib

Figure 98 present role class library “AutomationMLFMILogicRoleClassLib” as object tree. Within this role class library abstract and base role classes for the integration of FMU/FMI models are defined.

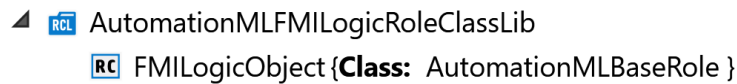


Figure 98: Overview AutomationMLFMILogicRoleClassLib as AutomationML tree

Note: The it is planned to migrate the role class library into [WP-Part4:2018] within the next maintenance cycle of the IEC standard representation of the whitepaper.

4.1.4.1 RoleClass FMILogicObject

The role class “FMILogicObject” shall be used as specified in Table 67.

Table 67: RoleClass FMILogicObject

Class name	FMILogicObject
Description	The role class “FMILogicObject” shall be used to integrate FMI based objects into AutomationML. It is a basic abstract role class.
Parent class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole
Path for element reference	AutomationMLFMILogicRoleClassLib/FMILogicObject
Attributes	None
Interfaces	None

4.2.1 Overview

The interface classes are derived from role classes defined from AutomationML basic interfaces defined in [WP-Part1:2018].

```

classDiagram
    package AutomationMLBPRInterfaceClassLib {
        class ExternalDataReference {
            <<InterfaceClass>>
        }
    }

    package AutomationMLInterfaceClassLib {
        class AutomationMLBaseInterface {
            <<InterfaceClass>>
        }
        class ExternalDataReference {
            <<InterfaceClass>>
        }
    }

    package AutomationMLFMIInterfaceClassLib {
        class FMIVariableInterface {
            <<ExternalInterface>>
        }
        class FMReference {
            <<ExternalInterface>>
        }
    }

    package AutomationMLComponentBaseICL {
        class 2DReference {
            <<InterfaceClass>>
        }
        class GraphicalRepresentationReference {
            <<InterfaceClass>>
        }
        class JTReference {
            <<InterfaceClass>>
        }
        class DeviceDescriptionReference {
            <<InterfaceClass>>
        }
        class MechanicInterface {
            <<InterfaceClass>>
        }
        class ElectricInterface {
            <<InterfaceClass>>
        }
        class PneumaticInterface {
            <<InterfaceClass>>
        }
        class PneumaticConnector {
            <<InterfaceClass>>
        }
        class CondensateDrainConnector {
            <<InterfaceClass>>
        }
        class SkillInterface {
            <<InterfaceClass>>
        }
    }

    JointInterface <|-- DeviceDescriptionReference
    SesorInterface <|-- MechanicInterface
    LiquidicInterface <|-- ElectricInterface
    HydraulicInterface <|-- SkillInterface

    AutomationMLBaseInterface <|-- ExternalDataReference
    
```

Figure74: AutomationMLComponentBase/CL

Figure 99 present the normative interface class library “AutomationMLComponentBaseICL” as object tree. Within this interface class library abstract and base interface role classes are defined. This interface classes are used to integrate the different model aspects of AutomationML Component or Composite Components and to interconnect them.
















- ▲  AutomationMLComponentBaseICL
 -  SkillInterface {**Class:** AutomationMLBaseInterface }
 -  2DReference {**Class:** ExternalDataReference }
 -  GraphicRepresentationReference {**Class:** ExternalDataReference }
 -  JTReference {**Class:** ExternalDataReference }
 -  DeviceDescriptionReference {**Class:** ExternalDataReference }
 -  MechanicInterface {**Class:** AutomationMLBaseInterface }
 -  ElectricInterface {**Class:** AutomationMLBaseInterface }
 -  LiquidicInterface {**Class:** AutomationMLBaseInterface }
 -  PneumaticInterface {**Class:** AutomationMLBaseInterface }
 -  PneumaticConnector {**Class:** PneumaticInterface }
 -  CondensateDrainConnector {**Class:** AutomationMLBaseInterface }
 -  HydraulicInterface {**Class:** AutomationMLBaseInterface }
 -  SensorInterface {**Class:** MechanicInterface }
 -  JointInterface {**Class:** AutomationMLBaseInterface }

Figure 99: Overview AutomationMLComponentBaseICL

4.2.2.1 InterfaceClass GraphicRepresentationReference

The interface class “GraphicRepresentationReference” shall be used as specified in Table 83

Table 68: InterfaceClass GraphicRepresentationReference

Class name	GraphicRepresentationReference	
Description	The interface class “GraphicRepresentationReference” is shall be used to reference a graphihic representation of the AutomationML Component or Composite Component.	
Parent class	AutomationMLBPRInterfaceClassLib/ExternalDataReference	
Path for element reference	AutomationMLComponentBaseICL/GraphicRepresentationReference	
Attribute	refURI (inherited)	This URI must point to a referenced graphic representation document.
	MIMETYPE (inherited)	MIMETYPE of the file according to [RFC2046:1996].

4.2.2.2 InterfaceClass 2DReference

The interface class “2DReference” shall be used as specified in Table 69

Table 69: InterfaceClass 2DReference

Class name	2DReference	
Description	The interface class “2DReference” is shall be used to reference a 2D representation of the AutomationML Component or Composite Component.	
Parent class	AutomationMLBPRInterfaceClassLib/ExternalDataReference	
Path for element reference	AutomationMLComponentBaseICL/Reference2D	
Attribute	refURI (inherited)	This URI must point to a reference to a 2D representation of the AutomationML Component or Composite Component.
	MIMETYPE (inherited)	MIMETYPE of the file according to [RFC2046:1996].

4.2.2.3 InterfaceClass JTReference

The interface class “JTReference” shall be used as specified in Table 70.

Table 70: InterfaceClass JTReference

Class name	JTReference	
Description	The interface class “JTReference” is shall be used to reference a JT model representation of the AutomationML Component or Composite Component.	
Parent class	AutomationMLBPRInterfaceClassLib/ExternalDataReference	
Path for element reference	AutomationMLComponentBaseICL/JTReference	
Attribute	refURI (inherited)	This URI must point to a JT file representing the AutomationML Component.
	MIMETYPE (inherited)	MIMETYPE of the file according to [RFC2046:1996].

4.2.2.4 InterfaceClass SkillInterface

The interface class “SkillInterface” shall be used as specified in Table 71.

Table 71: InterfaceClass SkillInterface

Class name	SkillInterface	
Description	The interface “SkillInterface” shall be required for linking a component skill information.	
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface	
Path for element reference	AutomationMLComponentBaseICL/SkillInterface	
Attribute	None	

4.2.2.5 InterfaceClass DeviceDescriptionReference

The interface class “DeviceDescriptionReference” shall be used as specified in Table 72.

Table 72: InterfaceClass DeviceDescriptionReference

Class name	DeviceDescriptionReference	
Description	The interface class “DeviceDescriptionReference” is an abstract basic interface class and the base class for standard or user defined interface classes referencing a technology based device description file.	
Parent class	AutomationMLBPRInterfaceClassLib/ExternalDataReference	
Path for element reference	AutomationMLComponentBaseICL/DeviceDescriptionReference	
Attribute	Version xs:string	The version of the actual device description file that is referenced here.
	refURI (inherited)	This URI must point to a technology related device description file according to the respective technology standard.
	“MIMEType” (inherited)	MIMEType of the file according to [RFC2046:1996].

4.2.2.6 InterfaceClass MaintenanceDescriptionLink

The interface class “MaintenanceDescriptionLink” shall be used as specified in Table 73.

Table 73: InterfaceClass MaintenanceDescriptionLink

Class name	MaintenanceDescriptionLink
Description	The interface “MaintenanceDescriptionLink” shall be used to interlink MaintenanceDescriptionGroup and MaintenanceDescriptionItem.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/MaintenanceDescriptionLink
Attribute	None

Note: The InterfaceClass is delayed in version 1.0.1

4.2.2.7 InterfaceClass MechanicInterface

The interface class “MechanicInterface” shall be used as specified in Table 74.

Table 74: InterfaceClass MechanicInterface

Class name	MechanicInterface
Description	A “MechanicInterface” represents a hardware provision to mechanically fasten or join two or more objects together.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/MechanicInterface
Attribute	None

4.2.2.8 InterfaceClass ElectricInterface

The interface class “ElectricInterface” shall be used as specified in Table 75.

Table 75: InterfaceClass ElectricInterface

Class name	ElectricInterface
Description	An “ElectricInterface” describes an electro-mechanical provision used to join electrical terminations and to create electrical circuits.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/ElectricInterface
Attribute	None

Note: A derived InterfaceClassLib shall be provided that defines specific automation interface types, e.g. M12, M8, M5, RJ45, 7/8inch. Exempels for the use can be found

4.2.2.9 InterfaceClass LiquidicInterface

The interface class “LiquidicInterface” shall be used as specified in Table 76

Table 76: InterfaceClass LiquidicInterface

Class name	LiquidicInterface
Description	A “LiquidicInterface” is an interface to describe a liquidic connection between to automation components.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/LiquidicInterface
Attribute	None

4.2.2.10 InterfaceClass PneumaticInterface

The interface class “PneumaticInterface” shall be used as specified in Table 77.

Table 77: InterfaceClass PneumaticInterface

Class name	PneumaticInterface
Description	A “PneumaticInterface” is an interface to describe a pneumatic connection point of an automation component in general. It can be used to join pneumatic components and create pneumatic circuits.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/PneumaticInterface
Attribute	None

4.2.2.11 InterfaceClass PneumaticConnector

The interface class “PneumaticConnector” shall be used as specified in Table 77.

Table 78: InterfaceClass PneumaticConnector

Class name	PneumaticConnector	
Description	A “PneumaticConnector” is a PneumaticInterface that represents a pneumatic connector standardized by [ISO 18582-2].	
Parent class	AutomationMLComponentBaseICL/PneumaticInterface	
Path for element reference	AutomationMLComponentBaseICL/PneumaticConnector	
Attribute	pneumaticPort xs:string	Type and size of the pressure media connection on a pneumatic automation component. All values from the property “pneumatic port” from [ISO18582-2] are valid
	connectorType xs:string	Type of the use of the connection, following [ISO18582-2], but the relevant attributes (“pneumatic input port”, “pneumatic output port”, “pneumatic exhaust port”, “pneumatic pilot ort”) are summed up here to one AutomationML attribute. The following values are accordingly allowed: (none) input output exhaust pilot

4.2.2.12 InterfaceClass CondensateDrainConnector

The interface class “CondensateDrainConnector” shall be used as specified in Table 77.

Table 79: InterfaceClass CondensateDrainConnector

Class name	CondensateDrainConnector
Description	A “CondensateDrainConnector” is an interface to describe a connector for the condensate drain connection of pneumatic automation component. It is standardized by [ISO18582-2].
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/CondensateDrainConnector
Attribute	None

4.2.2.13 InterfaceClass HydraulicInterface

The interface class “HydraulicInterface” shall be used as specified in Table 80.

Table 80: InterfaceClass HydraulicInterface

Class name	HydraulicInterface
Description	A “HydraulicInterface” is an interface to describe a hydraulic connection point of an automation component. It can be used to join hydraulic components and create hydraulic circuits.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/HydraulicInterface
Attribute	None

4.2.2.14 InterfaceClass SensorInterface

The interface class “SensorInterface” shall be used as specified in Table 80.

Table 81: InterfaceClass SensorInterface

Class name	SensorInterface
Description	A “SensorInterface” is an process interface connect the component to physical properties of interest. It will be specified in detail in version 2 of the document.
Parent class	AutomationMLComponentBaseICL/MechanicInterface
Path for element reference	AutomationMLComponentBaseICL/SensorInterface
Attribute	None

4.2.2.15 InterfaceClass JointInterface

The interface class “JointInterface” shall be used as specified in Table 80.

Table 82: InterfaceClass JointInterface

Class name	JointInterface
Description	A “JointInterface” is an interface that connects a joint of an COLLADAKinematicJoint with other interfaces that have an influence, dependency or relevance to the joint.
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface
Path for element reference	AutomationMLComponentBaseICL/JointInterface
Attribute	None

4.2.3 AutomationMLFMIInterfaceClassLib

Figure 100 present role class library “AutomationMLFMIInterfaceClassLib” as object tree. Within this interface class library base external interface classes for the integration of FMU/FMI models are defined.

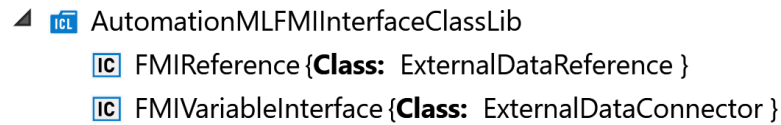


Figure 100: Overview AutomationMLFMIInterfaceClassLib

Note: The it is planned to migrate the interface class library into [WP-Part4:2018] within the next maintenance cycle of the IEC standard representation of the whitepaper.

4.2.3.1 InterfaceClass FMIRreference

The interface class “FMIRreference” shall be used as specified in Table 83

Table 83: InterfaceClass FMIRreference

Class name	FMIRreference	
Description	The interface class “FMIRreference” shall be used to reference an FMI model representation of the AutomationML Component or Composite Component.	
Parent class	AutomationMLBPRInterfaceClassLib/ExternalDataReference	
Path for element reference	AutomationMLFMIInterfaceClassLib/FMIRreference	
Attribute	refURI (inherited)	This URI must point to a references a Functional Mockup Unit (FMU) according to the FMI standard.
	MIMETYPE (inherited)	MIMETYPE of the file according to [RFC2046:1996].

4.2.3.2 InterfaceClass FMIVariableInterface

The interface class “FMIVariableInterface” shall be used as specified in Table 84

Table 84: InterfaceClass FMIVariableInterface

Class name	FMIVariableInterface	
Description	The interface class “FMIVariableInterface” references a variable of a co-simulation Functional Mockup Unit (FMU) according to the FMI standard (see [FMI:2019]) via its Name attribute. An FMU is a zipped file archive and the root contains a file called modelDescription.xml, which declares all available variables.	
Parent class	AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector	
Path for element reference	AutomationMLFMIInterfaceClassLib/FMIVariableInterface	
Attribute	Name xs:string	The variable name within the FMU declaration file modelDescription.xml. It must math a character string defined by the XPath /fmiModelDescription/ModelVariables/ScalarVariable@name.
	Causality xs:string	The causality defined for the variable within the FMU declaration file modelDescription.xml. It must math a character string defined by the XPath /fmiModelDescription/ModelVariables/ScalarVariable@causality. Especially the causality notifiers “input” and “output” will have value for AutomationML-based tools for connecting multiple simulation models.
	refURI (inherited)	The inherited attribute shall not be used for external interfaces of the interface class FMIVariableInterface..

5 Exchange of AutomationML Components as AMLX Container

This chapter describes the different types of AMLX Container, how they are internally organised and the file and folder naming conventions to be used. Additionally, it will be explained for which use cases the single AMLX Container support.

In general, all provisions of [BPR AMLX] shall be applied.

5.1 Use Cases for AMLX Container

Basically, three use cases are supported in the context of exchanging AutomationML Components and AutomationML Composite Components. These use cases are:

- First, sending of a single AutomationML Component,
In this use case the AMLX Container can be used to transfer a single component, even if it does not cover all aspects or if the AutomationML Component definition in this paper or is under development. A sub case of this is the sending of an AutomationML Composite Component. In this case no additional rules regarding name conventions within the AMLX Container, rules regarding mandatory or models exist.
- Second, sending libraries of AutomationML Components
In this use case a complete SUC library is packed into an AMLX Container. This can contain several components and their associated files. In this use case no additional rules regarding name conventions within the AMLX Container or mandatory and attributes exist.
- Third, sending of components specified in detail that meet a certain profile.
In this use case the AutomationML Components within the AMLX Containers have to follow additional rules regarding naming conventions, mandatory attributes, required models or the internal structure, that are defined in so-called profiles. An example for the usage of this use case are AutomationML Components that shall be used as device description. Such an AMLX Container can be certified according to these profiles. A first profile for such a profile will be defined within an extension of this whitepaper.

All these are useful in combination, e.g. for the application of an electronic product catalogue or future twin stores.

5.2 AMLX Container types for AutomationML Components

5.2.1 AMLX Container for single AutomationML Component

The AMLX Container for a single component shall contain a single SUC library containing a single SUC and the associated sub model data according to the following rules:

- The AMLX Container for a single component shall contain model type specific information of the single component type.
- The AMLX Container shall contain a SUC library with an AutomationML Component as single SystemUnitClass (SUC) and, if needed, further files which model the considered component. It may contain folders and subfolders containing further component related data and to allow structuring the files if needed.
- The SystemUnitClass contains information about the identification of the component, its external interfaces and further sub models according to this whitepaper
- The model files referenced from the SUC can be stored within the AMLX Container or outside of the AMLX container and e.g. referenced via stable links in the internet. It is however recommended to use self-contained packages whenever feasible and only reference to externally versioned standard libraries.
- The AMLX Container can contain the used AutomationML libs.

This AMLX Container type supports the first and third use case of this subchapter. Figure 109 shows such a minimum AMLX Container for a single AutomationML Component.

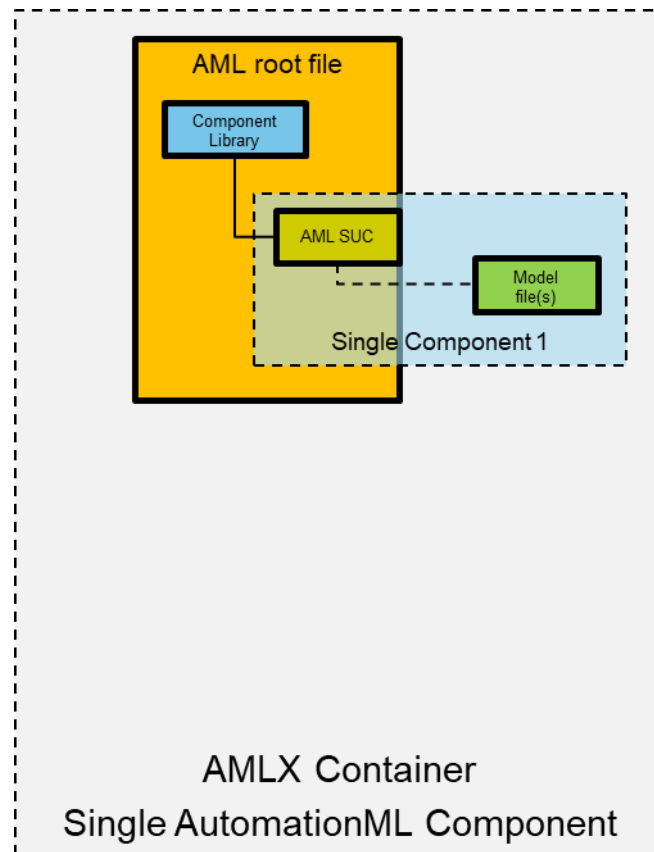


Figure 101: Minimum AMLX Container for a single component

5.2.2 AMLX Container for AutomationML Component catalogues

An AMLX Container containing multiple AutomationML Components of component types as catalogue shall be organized in the same manner as described above for the single component catalogue but can contain multiple components as system unit classes of the same vendor or multiple vendors. The AutomationML files shall model vendor information of the component types according to IEC62714.

This AMLX Container type supports the second use case of this subchapter. Figure 102 shows such an AMLX Container for a AutomationML Component catalogue.

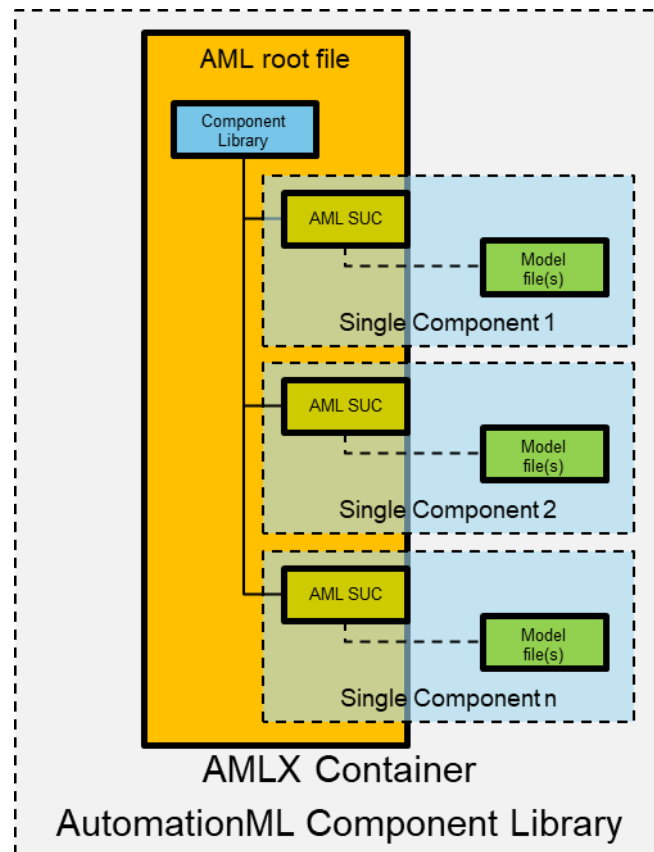


Figure 102: Minimum AMLX Container for a AutomationML Component catalogue

5.2.3 AMLX Container for single AutomationML Composite Component

An AMLX Container containing an AutomationML Composite Component type (synonyms: “combined component”, “unit”, “system of components”) requires modelling the subcomponents and their relations.

- An AMLX Container for a AutomationML Composite Component type shall be organized in the same manner as described above for the multiple component type as SUC library.
- The AutomationML Composite Component type shall be modelled as a single system unit class that aggregates multiple AutomationML Components as InternalElements.
- The sub AutomationML Component shall be an instance of an AutomationML Component type definition.
- The AutomationML Composite Component shall model its type specific information and the interrelations to subcomponents.
- Each sub AutomationML Component shall be modelled as additional system unit classes. This may result in multiple component catalogues.
- Sub AutomationML Components within the same AMLX Container shall be modeled in the same system unit class lib or in a system unit class lib that is part of the root file. Nesting is only allowed in the SUC's InternalElement hierarchy.
- All subcomponent types may reference external files within or outside the AMLX Container.

This AMLX Container type supports the first and third use case of this subchapter. Figure 103 shows such an AMLX Container for an AutomationML Composite Components.

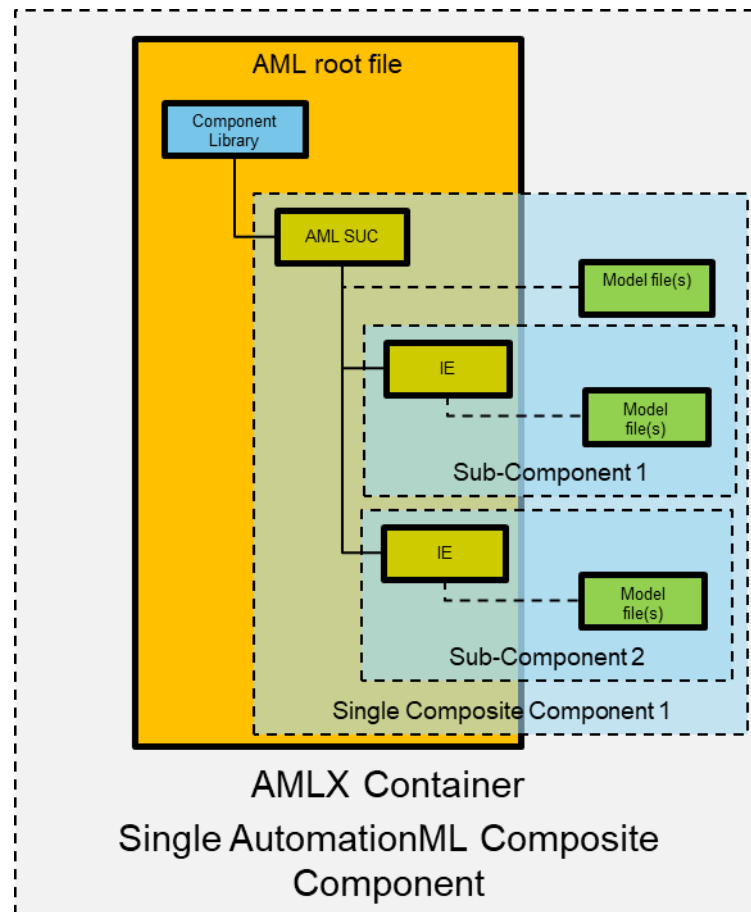


Figure 103: Minimum AMLX Container for an AutomationML Composite Component

6 Manufacturer Specific Extensions of Component Description

To use the previously described role Classes for modelling manufacturer specific component descriptions the following steps are recommended.

In a first step the “AutomationMLComponentBaseRCL” and the “AutomationMLComponentStandardRCL” shall be checked if the predefined roles can already fulfill the requirements of manufacturer. Only if the requirements cannot be covered by the standard role classes the manufacturer will define own roles. To this aim the already predefined role classes of the “AutomationMLComponentBaseRCL” and the “AutomationMLComponentStandardRCL” role class libraries shall be used. Hereby must be considered that special models (geometry, simulation, behaviour) are already defined. Only when no suitable roles can be found in the libraries the manufacturer will define own, new role classes in a separate manufacturer-specific role class library.

In the same way the interfaces of the components shall be defined. Only when no suitable interfaces can be found in the “AutomationMLComponentBaseICL” the manufacturer will define own, new interface classes in a separate manufacturer-specific interface class library.

But all new role or interface classes shall be derived from the role classes of the “AutomationMLComponentBaseRCL” and the “AutomationMLComponentStandardRCL” role class lib or the “AutomationMLComponentBaseICL” interface class library.

In a final step the system unit classes for the engineering domain can be identified and modelled as templates for further use. Here the structure of the components can be modelled especially with respect to the relevant properties to be considered. Therefore, appropriate internal elements and attributes can be added.

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

Chapter 9.5 shows an example for the definition of a component using a standard behaviour model (PLCopen XML) and the definition of a component using a manufacturer specific behaviour model (SIMIT) as well.

7 Semantics in Automation Components

One critical point within the modelling and description of automation components as AutomationML Components is the correct interpretation of the component data. One major problem is that the incoming data point need to be identifiable automatically by the receiving tool.

A methodology to handle this problem is to define the syntax of data objects and to the integration of a semantic specification, i.e. each object will carry its own semantics definition. The semantic definition carried within the data object has to be unique. Therefore, appropriate means have to be available.

Within the industry, several catalogue standards are available defining object and property semantics uniquely. Usually they follow the IEC 61360 standard. Examples are the eCI@ss catalogue standard and the IEC Common Data Dictionary standard. The following section introduces the concept of how a semantic system is integrated into the AutomationML Component description.

Note: The usage of semantic system or attribute/property catalogues shall always follow the licences terms of publishing organisation.

7.1 Concept of Semantic System Integration

The integration of semantic systems based on two aspects. The first aspect is that the used semantic systems will be announced for each AutomationML Component whether it is a standardized or proprietary system. The second approach is to reference the semantic definition of the semantic systems to single attributes.

7.1.1 Definition of used Semantic Systems

To announce the use of a semantic system for an AutomationML Component a role class shall be used. This role class is defined as “AutomationComponentSemanticSystem” in Table 26. The role class contains five complex attributes. One is as generic attribute to announce the use of an own semantic systems and four are predefined systems that are supported from the AutomationML Component definition. The supported semantic systems are the classification systems [IEC62683-SC3D:2014], [IEC 61360-4:2005], [IEC 61987:2016] and [CI@ss:2020]. The structure of the role class is depicted in Figure 104.

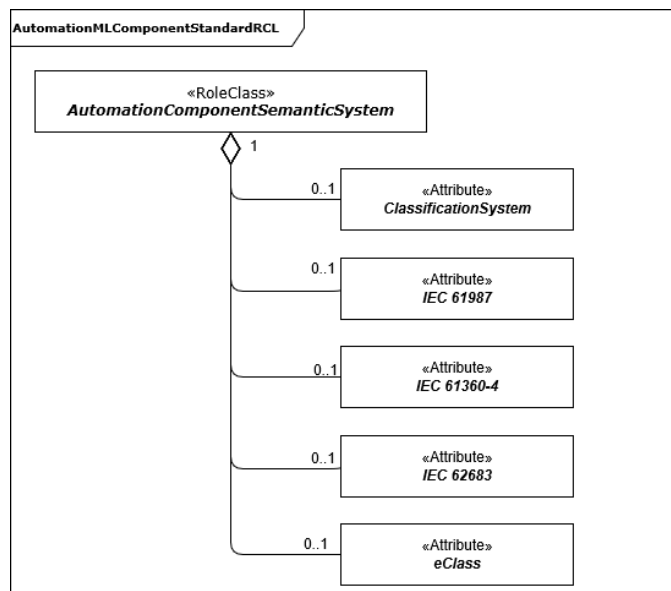


Figure 104: UML Representation of role class “AutomationComponentSemanticSystem”

Each of the complex attributes has the sub attributes “Version”, “RefSemanticPrefix” and “URL”. These sub attributes are used to define information about the integrates semantic system. Additional contains the attribute “ClassificationSystem” the sub-attribute “Name”.

For the use of the role class the following rules shall apply additionally:

- If a semantic system or classification system shall be announced for an AutomationML Component the root of the AutomationML Component shall reference the role class "AutomationComponentSemanticSystem".
- The attributes shall be used as specified in Table 26, Table 27 and Table 28.

7.1.2 Referencing Attributes Semantic

AutomationML standard allows industrial practitioners to interlink and integrate heterogeneous data more efficiently and support the transfer technical data for AutomationML Components or Composite Components from one company to another company without any data loss. In order to transmit error free data or to thoroughly integrate all complex data of any object, interlinking with correct meaning is required. Therefore the AutomationML concept of reference semantic is used.

With the help of reference semantic (RefSemantics) a particular semantic standard can be assigned to an attribute so that data which has to be transmitted is not miss interpretable.

But also, further sources of semantics shall be possible ranging from dictionaries defined by industrial associations like VDMA or VDA up to company based application recommendations of data objects.

Figure 105 shows the use of RefSemantic in AutomationML.

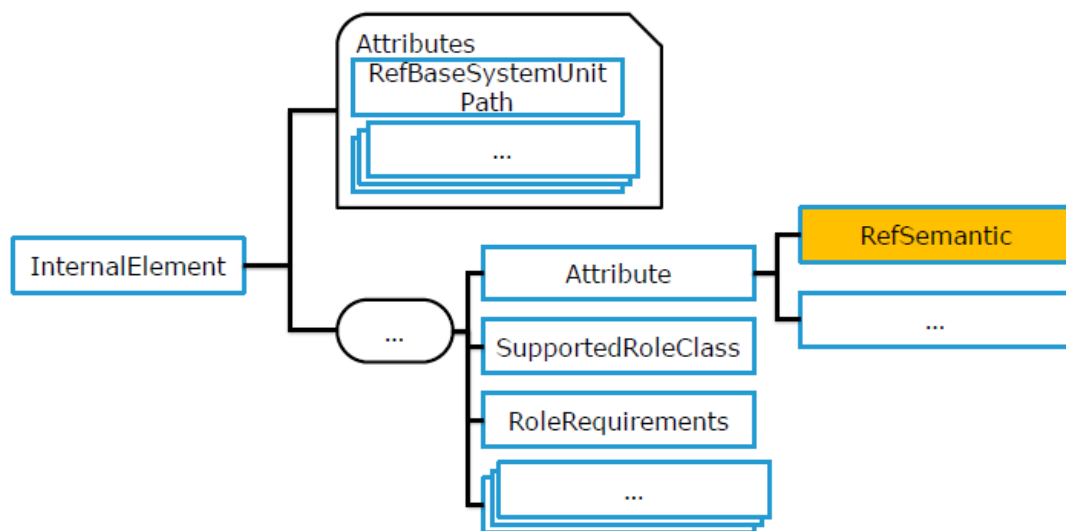


Figure 105: RefSemantic as AutomationML element

Additional it is necessary to support different semantic systems to specify attributes of AutomationML Components or Composite Components. Therefore, these systems have to be announced within the AutomationML Component description. This shall be done by using the role class "AutomationMLComponentSemanticSystem", see chapter 7.1.1.

For the definition of attribute semantics following shall apply:

- The CAEX schema element "RefSemantic" shall be used for the semantic definition for a single attribute, see Figure 105.
- The value of the XML attribute "CorrespondingAttributePath" of the CAEX schema element "RefSemantic" element shall be assembled following the URI structure defined in [RFC3986:2005] exploiting the elements scheme, path, query, and fragment. The scheme shall identify the source of semantics like an IRDI following eCI@ss or IEC CDD or a dedicated company. Next the path element is delimited from the source by "/" and indicates the location of the semantics definition depending on the scheme. After an "?" as delimiter the query can follow giving an additional interpretation to the named semantics definition. Finally, after a "#" delimiter the object identifier of the semantics definition in the semantics definition files is named.

Scheme as semantic owner // **Path** to semantics location ? optional **Query** to improve semantic uniqueness # **Fragment** as ID within the semantic system

Note 1: If a standard ID like IRDI exist within the semantic system this shall be used.

Note 2: Examples for the “CorrespondingAttributePath” can be found in Figure 106.

- Scheme shall identify the semantic owner.
- Path and Query shall uniquely identify the semantic system.
- The ID within the semantic system shall be used as defined within the system.
- Each used semantic system that is used within one AutomationML Component or Composite Component definition shall be announced following the specification of the role class “AutomationMLComponentSemanticSystem”.

myScheme://myfilepath/myfile.mimetype?myQuery#myobjectID

IRDI-PATH://0173-1#002-BAA018#004

IRDI-PATH://0112/2///61360_4#ABC484#003

myCompany://SemanticSystem#SemanticElementID

Figure 106: Examples for CAEX attribute “CorrespondingAttributePath”

An example of the use of the semantic system integration can be found in chapter 9.8.

7.2 Example for Usage

Within this sub chapter two example for the usage of the semantic system integration are presented. The first example shows an AutomationML Component that uses the [IEC61987] as semantic system. In the second example an AutomationML Component will use an own semantic system to specify the semantic of attributes.

Example: AutomationMLComponentIEC 62683

Within the example an AutomationML Component with the name “AutomationMLComponentIEC_62683” is defined as system unit class. This system unit class has two supported role classes, see, Figure 107:

- The role class “AutomationComponent” to indicate that the system unit class represents an AutomationML Component.
- The role class “AutomationMLComponentSemanticSystem” to indicate that the AutomationML Component supports a semantic system.

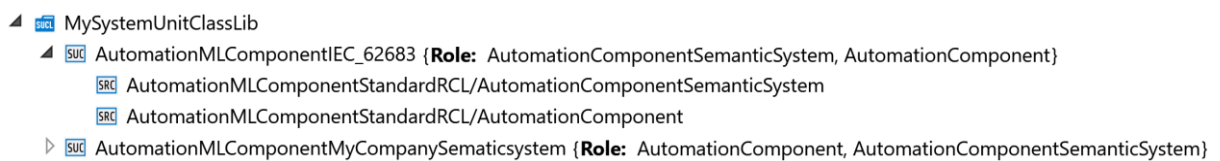


Figure 107: Examples AutomationML Component with standard semantic system

Within the example the definitions of the IEC 62683 [IEC62683-SC3D:2014] are used to define the semantic of the attributes of the AutomationML Component. The use of the semantic system is defined with the used of the predefined attribute “IEC 62683”, see Figure 106.

The use of the semantic system is show within the attribute “Manufacturer”. This attribute has the value “MyCompany” and additional ther value “IRDI:0112/2///62683#ACE102#001” for the RefSemantic. The IRDI references the IEC 62683 property “manufacturer name”² with all definitions for this property.

Name	Value	Data Type	Semantic
IEC 62683		xs:string	
Version	V2.0014.0016	xs:string	
RefSemanticPrefix	IRDI:0112/2///62683#	xs:string	
URL	https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/TreeFrameset?OpenFrameSet&ongletactif=1	xs:string	
IdentificationData		Empty	IRDI:0112/2///62683#ACC011#001
Manufacturer	MyCompany	xs:string	IRDI:0112/2///62683#ACE102#001

Figure 108: Examples usage of semantic systems for attribute definition

² <https://cdd.iec.ch/cdd/iec62683/iec62683.nsf/TreeFrameset?OpenFrameSet&ongletactif=1>

Example: AutomationMLComponentMyCompanySemanticSystem

Within the example an AutomationML Component with the name “AutomationMLComponentMyCompanySemanticSystem” is defined as system unit class. This system unit class has two supported role classes, see, Figure 109:

- The role class “AutomationComponent” to indicate that the system unit class represents an AutomationML Component.
- The role class “AutomationMLComponentSemanticSystem” to indicate that the AutomationML Component supports a semantic system.

```

MySystemUnitClassLib
└─ AutomationMLComponentMyCompanySemanticsystem {Role: AutomationComponent, AutomationComponentSemanticSystem}
   └─ AutomationMLComponentStandardRCL/AutomationComponent
   └─ AutomationMLComponentStandardRCL/AutomationComponentSemanticSystem

```

Figure 109: Examples AutomationML Component with own semantic system

Within the example an own semantic system is used to define the semantic of the attributes of the AutomationML Component. The name and all properties of this system are defined within the sub attributes of the attribute “ClassificationSystem”, see Figure 112. The name of the semantic system is “MyCompanySystem”, it is used in version “1.0” and it is referenced with the prefix “myCompany://” within the RefSemantic block of the attributes.

The use of this own defined semantic system is show within the attribute “Manufacturer”. This attribute has the value “MyCompany” and attitional ther value “myCompany://Filetype=ElementID” for the RefSemantic.

Name	Value	DataType	Semantic
ClassificationSystem		xs:string	
Name	MyCompanySystem	xs:string	
Version	1.0	xs:string	
RefSemanticPrefix	myCompany://	xs:string	
IdentificationData		Empty	
Manufacturer	MyCompany	xs:string	myCompany://Filetype#ElementID

Figure 110: Examples usage of own semantic systems for attribute definition

8 Automation Components in CAEX 3.0

8.1 General

The component libraries are based on AutomationML version 2.0, which in turn is based on CAEX version 2.15. The next AutomationML version 2.10, which is based on CAEX version 3.0, is currently in the standardization process. At the time of writing this whitepaper, AutomationML part 1 version 2.10 is already standardized. For Part 2, which defines additional RoleClass libraries, the standardization process has just started.

When migrating the component libraries to the next higher AutomationML version 2.10, two things have to be distinguished. These are the pure schema transformation from CAEX Version 2.15 to CAEX Version 3.0 and the semantic transformation to AutomationML Version 2.10.

Within this chapter 8 general transformation rules will form to AutomationML version 2.0 will be described in section 8.2 and 8.3. In chapter 8.4 these rules will be applied to model electrical interfaces of automation components as AutomationML Components in AutomationML Version 2.10 with CAEX 3.0.

8.2 CAEX Schema Transformation

The CAEX schema transformation from version 2.15 to version 3.0 is supported by the current version of the AutomationML editor. No information is lost during an upward transformation. With regard to the component libraries, this means that only some administrative changes are made, such as the transformation of the WriterHeader object to the SourceDocumentInformation and the insertion of the element SuperiorStandardVersion referring to AutomationML 2.10 as well as the use of the namespace xmlns="http://www.dke.de/CAEX".

8.3 AutomationML Version Transformation

The semantic migration from AutomationML 2.0 to AutomationML 2.10 is more complex. So far there is no tool support for this but it has to be done manually.

The transformation can take place in two steps. The first step involves the exchange of the used libraries whose objects are used in the component libraries. The second step involves the adaptation of the component libraries.

8.3.1 Exchange of used libraries

This step requires, that all used libraries are available in AutomationML version 2.10. This means that the transformation can only take place when all libraries used have been transformed. Currently the part 1 libraries, which are the AutomationMLBaseRoleClassLib and the AutomationMLInterfaceClassLib are ready to use. An additional AttributeType library has been added in version 2.10, where the attributes used in part 1 are globally defined. This AttributeType library will be extended with attribute definitions from part 3 and part 4. Part 2 doesn't define attribute types.

When adapting the objects used from external libraries, proceed as follows:

1. Check, if the CAEX path of the used object is still valid. For example, the role class ExternalData and the interface class ExternalDataReference, which were not standardized in AutomationML version 2.0 but are described in a recommendation, are now included in the basic libraries of version 2.10. The reference paths to these classes have to be modified.
2. Check, if a referenced element still exists. Some role classes, which exist in version 2.0 have been removed in version 2.10 like for instance the role class ,Port' and the role class ,PropertySet'. These removed role classes have to be substituted by their changed modelling concept.

8.3.2 Adaptation of the component libraries

To make the component libraries compatible with AutomationML 2.10, it is necessary to revise the defined attributes. It is advisable to store the attribute definitions in an AttributeType library and to enter references to the corresponding attribute type in all places of use. You should also check whether

attributes are related to already defined attributes in dependent libraries and whether an inheritance relationship to those attribute types can be defined.

CAEX 3.0, which is the basis for AutomationML 2.10, allows the definition of InterfaceClasses which contain nested Interfaces. When adapting, it should therefore also be checked whether a defined interface class or a coordinated modeling concept can be better represented by a nested interface class.

8.4 Modelling of electrical interfaces with CAEX 3.0

8.4.1 Whats new

With CAEX 3.0, the electric connector model is splitted into its logic model and its physical model.

- With CAEX 2.15, the electric connector models the physical connector (e.g. M12) with its inner pins.
- With CAEX 3.0 (see Figure 111), the electric connector represents the abstract/logic connector (e.g. M12 Ethernet). This covers its applications, and its current application may be assigned dynamically dependent on the use case. The logic electric connector (e.g. M12 Ethernet) models its physical implementation (e.g. M12 A coded 4 Pin) by means of an ExternalInterface. The pins of the physical connector are then modelled as nested ExternalInterfaces of the physical connector.

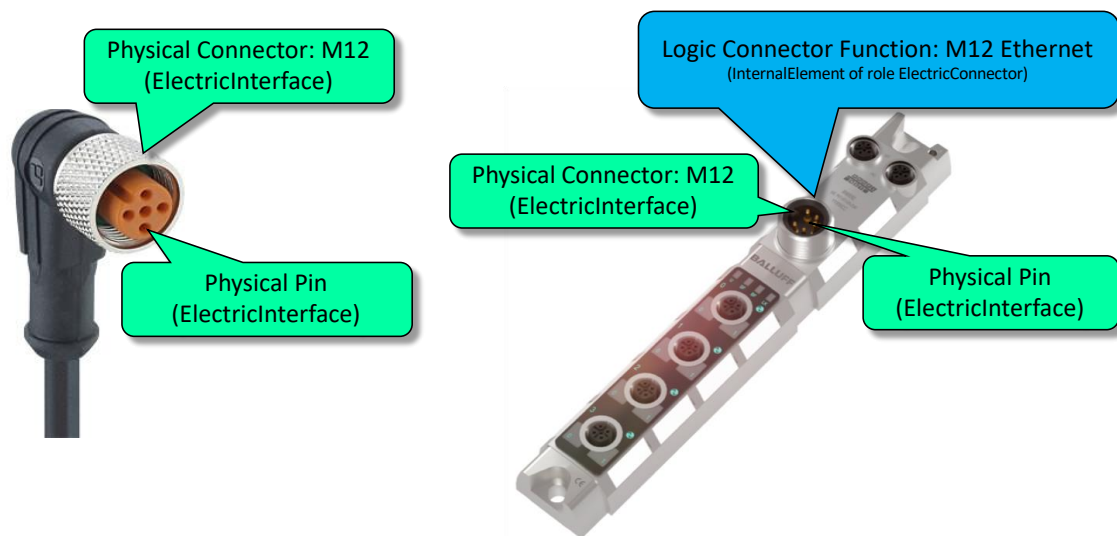


Figure 111: Logic connector versus physical connector versus physical interface (pin)

Technically, the modelling based on InterfaceClasses provides a number of benefits:

- Interfaces can be nested, hence modelling of complex nested interfaces is possible.
- Electric interfaces can be directly interlinked bypassing the need of interconnecting all sub pins. This is useful when the mechanics implicitly interlinks the pins when the connectors are connected. This simplifies models and reduces the model size.
- The electric connector application and its physical implementation is separated.
- Standard interface classes can be used as templates. This increases the value of standardizing electric connector libraries.

This chapter exemplarily illustrates the modelling of electrical interfaces by means of M12 connectors according to IEC61076.

The general modelling principles of chapter 3.13.4 shall apply for CAEX 3.0 as well except the provisions above.

8.4.2 Example: M12 interface class library

The M12 connectors are standardized in IEC61076 and exist in multiple variants with different geometry (coding), number of pins, use cases and properties. An example M12 connector is A coded with 4 pins.

The coding of M12 means a certain geometry, providing that only M12 connectors with the same coding fit together.

An M12 contains sub-pins and hence has a nested characteristics. Since CAEX 3.0 supports modelling of nested interfaces, all M12 interfaces are modelled as interface classes.

Figure 159 shows an interface class library *ConnectorLib_IEC61076*, a closer description of the classes is given in. The library demonstrates the general modelling principles. The full model of this library will be published in a separate white paper.

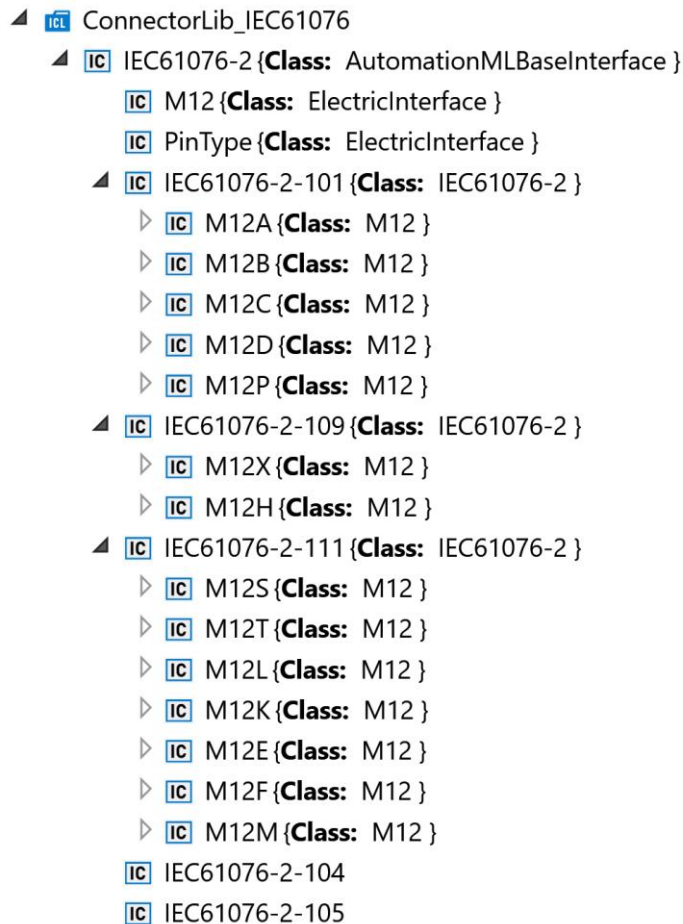


Figure 112: AML interface class library modelling a subset of possible M12 interfaces according to IEC61076-2

Class (of type)	Description
IEC61076ConnectorLib (library)	AML Interface ClassLibrary modelling M12 connectors according to IEC61076
IEC61076-2	Abstract InterfaceClass for the norm part 2 of IEC61076-2
M12 (ElectricInterface)	base class for the M12 connector, derived by ElectricInterface according to the Automation Component model
PinType (ElectricInterface)	InterfaceClass modelling an electric pin
IEC61076-2-101	Abstract InterfaceClass for IEC61076-2-101
M12A (M12)	Generic A coded M12 connector
M12B (M12)	Generic B coded M12 connector
M12D (M12)	Generic D coded M12 connector

M12P (M12)	Generic P coded M12 connector
M12X (M12)	Generic X coded M12 connector
IEC61076-2-109	Abstract InterfaceClass for IEC61076-2-101
M12X (M12)	Generic X coded M12 connector
M12H (M12)	Generic H coded M12 connector
IEC61076-2-111	Abstract InterfaceClass for IEC61076-2-101
M12S (M12)	Generic S coded M12 connector
M12T (M12)	Generic T coded M12 connector
M12L (M12)	Generic L coded M12 connector
M12K (M12)	Generic K coded M12 connector
M12E (M12)	Generic E coded M12 connector
M12F (M12)	Generic F coded M12 connector
M12M (M12)	Generic M coded M12 connector
IEC61076-2-104	Abstract InterfaceClass for IEC61076-2-104
IEC61076-2-115	Abstract InterfaceClass for IEC61076-2-105

A variety of M12 class attributes are explicitly modelled (see Figure 113) covering a wide range of engineering information which are generic for all variants of M12 connectors.

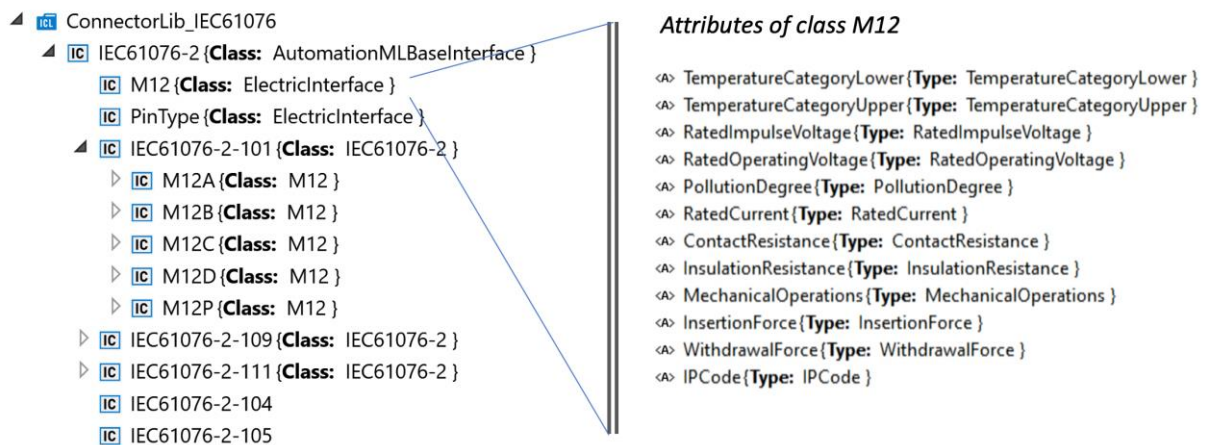


Figure 113: General attributes of the abstract M12 class, inherited to every M12 variant

The generic Pin type is modelled as InterfaceClass together with a set of attributes (see Figure 114).

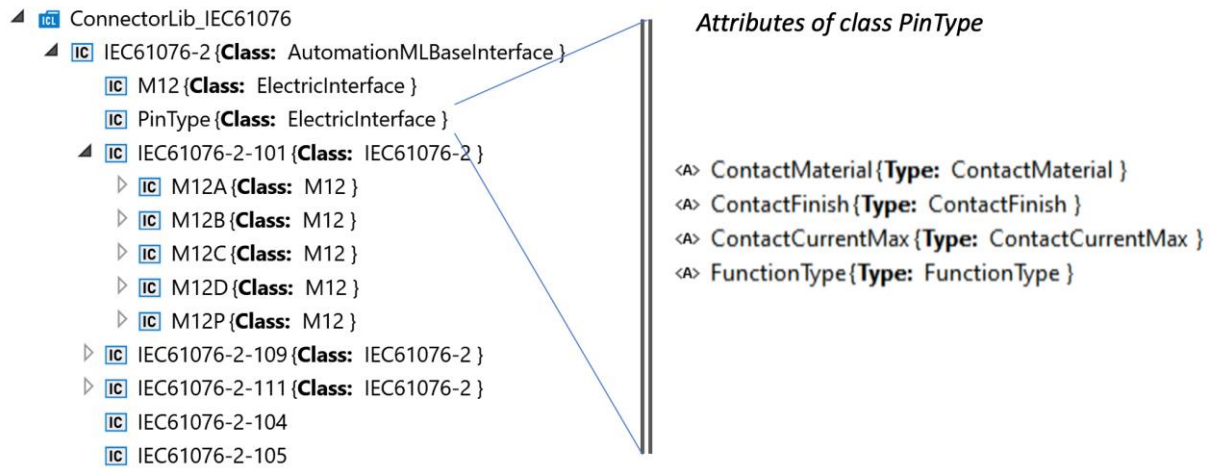


Figure 114: AML interface class library modelling a generic pin type

The AML model of a concrete M12 A coded interface with 4 pins is illustrated in Figure 115. The modeling starts with a 2 PIN Type A M12 interface. Then, the 3-Pin variant is derived from the 2-pin variant and the pin 3 is added. This is repeated until 5 pins and depicts that all A coded 2-5 Pin M12 connectors are physically identical and compatible among each other.

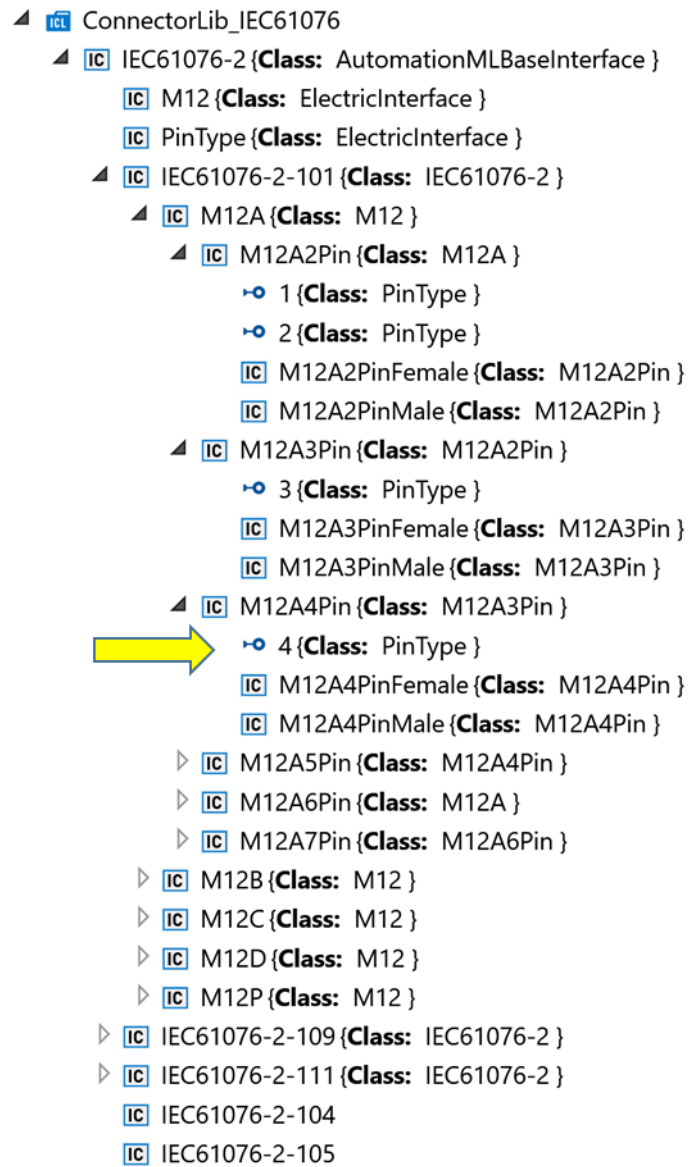


Figure 115: AML role class model of the M12 A coded with 4 pins and its male and female derivate

8.4.3 Example: Mini 7/8 interface class library

Figure 116 illustrates the modelling of Mini 7/8 inch electric connector types, which are based on ANSI B93.55M (NFPA T3.5.29M) plus reaffirmation notice 1988.

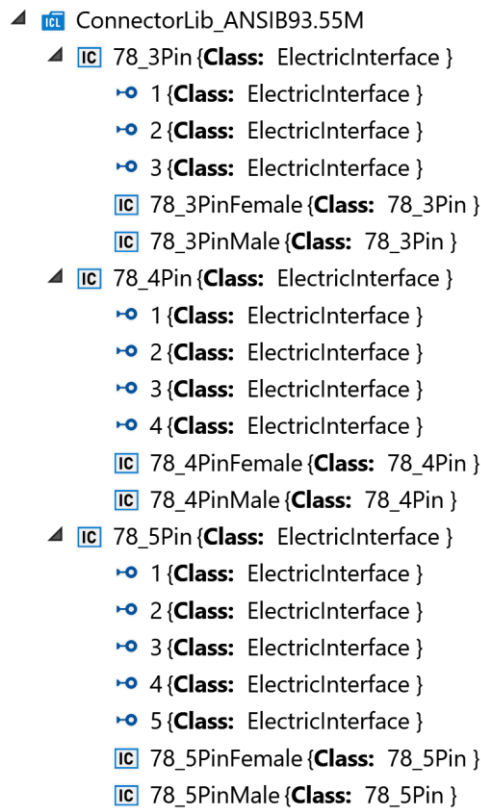


Figure 116: AML InterfaceClass library for Mini 7-8 interfaces

8.4.4 Example: RJ45 interface class library

RJ45 is standardized in IEC60603-7.

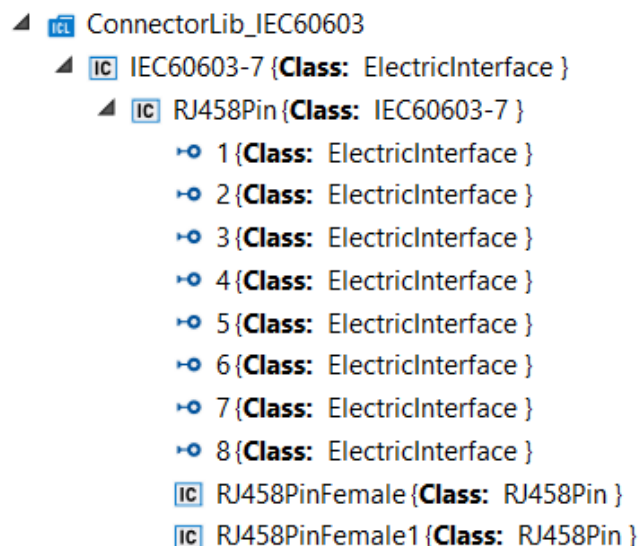


Figure 117: AML InterfaceClass library for an RJ45 interface

8.4.5 Application role class library

The first step is the modelling of generic connector functions as role library. This is shown in Figure 120.

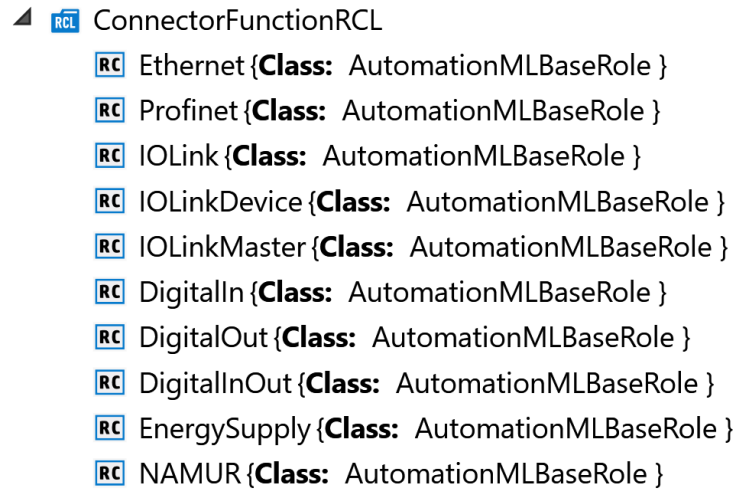


Figure 118: AML role class library modelling generic connector functions

8.4.6 Application Example: Automation component with multiple electric connectors

Figure 119 shows an example product catalogue modelled as SystemUnitClasslibrary. Utilizing the M12 role class library models an automation component "IOLinkMasterType123" with two M12 Ethernet connectors (A coded 5 Pins).

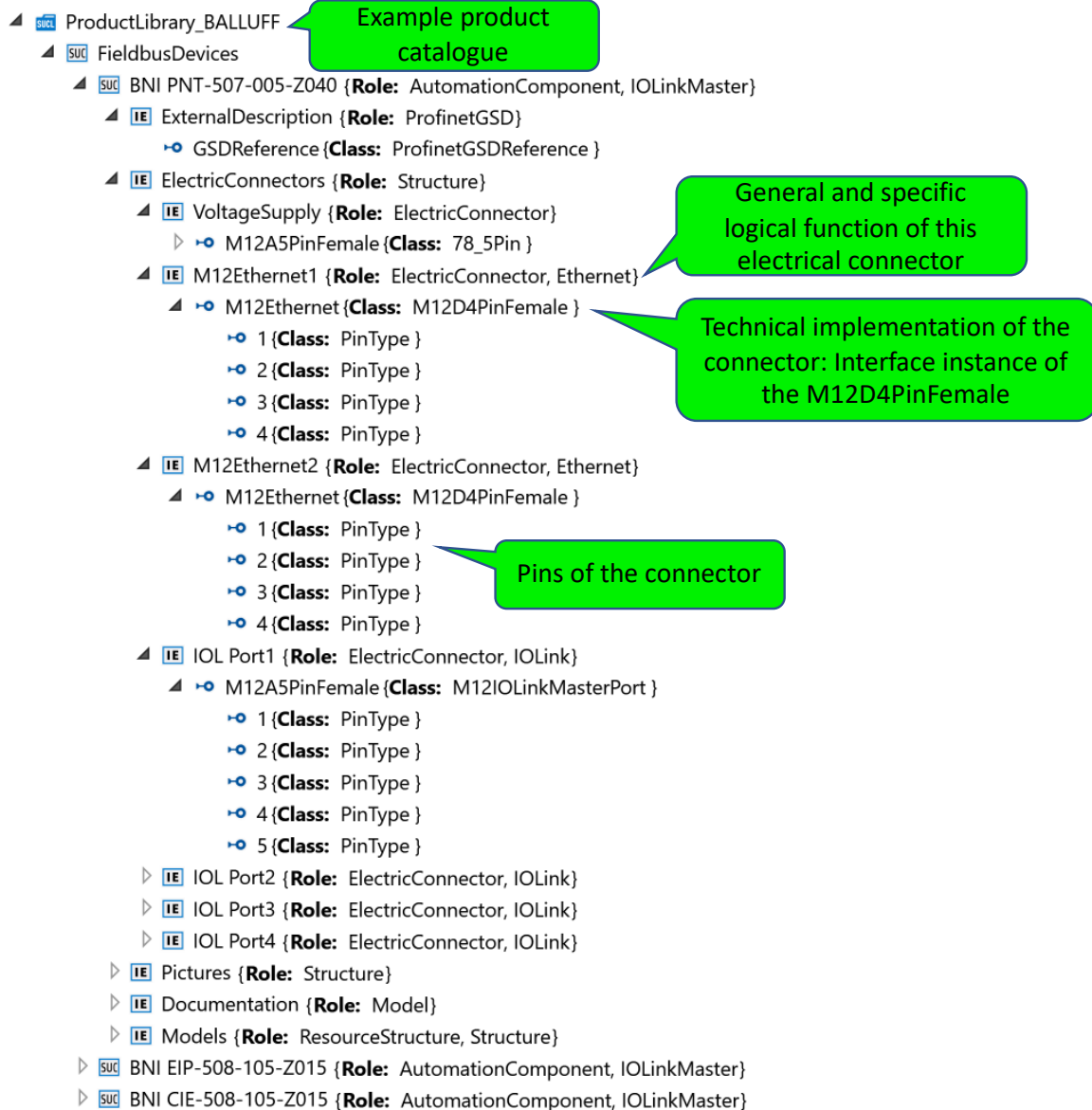


Figure 119: AML SystemUnitClass of an AutomationComponent with two M12 Ethernet connectors

8.4.7 Application Example: M12 to M12 cable

In this example, a cable with one M12 female and another M12 male connector is modelled and a modular modelling approach has been pursued. The modular modelling results in a CAEX System-UnitClass for a single wire, another SystemUnitClass for a cable with 4 wires, and a third System-UnitClass for the M12 to M12 cable which assembles all above classes.

Figure 120 assembles four of these single wires together in order to model a 4-wire-cable. Each single wire has own properties and inherits the interfaces.

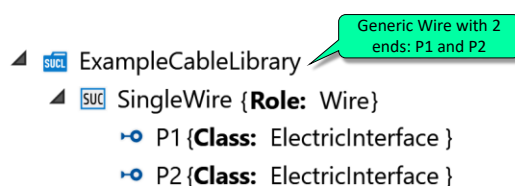


Figure 120: AML SystemUnitClass model of a single wire with two ends

Figure 121 assembles four of these single wires together in order to model a 4-wire-cable. Each single wire has own properties and inherits the interfaces.

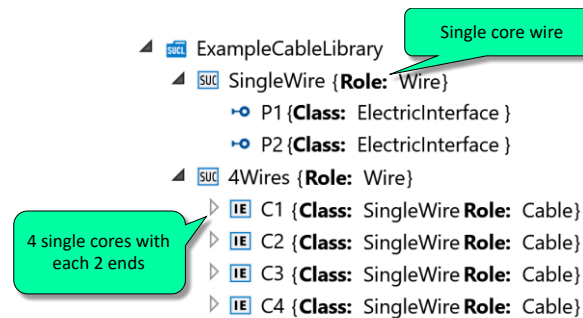


Figure 121: AML SystemUnitClass model of a cable with 4 wires

Figure 122 assembles the M12 to M12 cable (A-coded, 4 Pins). It consists of one 4-wire-cable, one M12 female interface and one M12 male interface. Furthermore, it models the wiring via CAEX InternalLinks.

What is interesting is that the modelling of the internal connections allows the modelling of any wiring: straight wiring, cross wiring and even wiring errors or cable fractures.

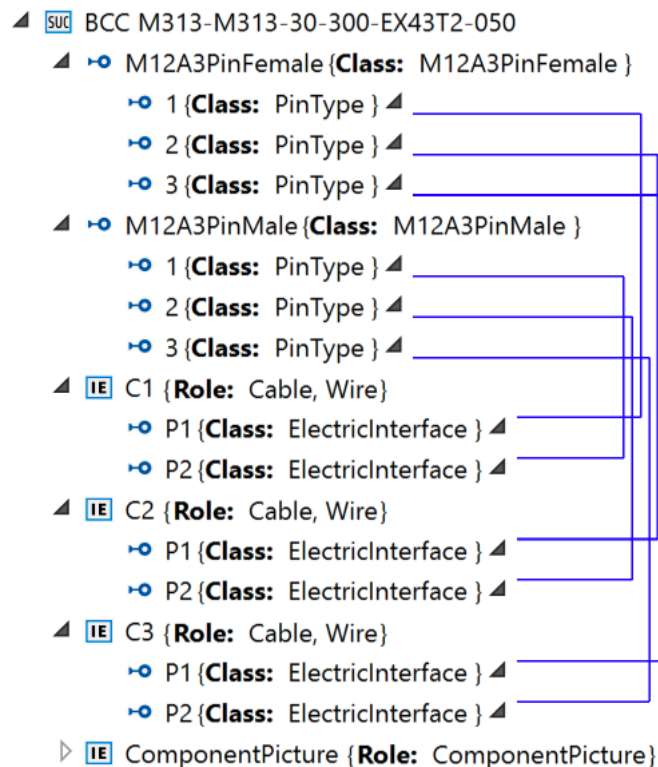


Figure 122: AML SystemUnitClass of an M12 to M12 cable with 3 wires

9 Practical examples for Automation Components

9.1 Example of a typical Automation Component: Pneumatic Cylinder

Within the following example an automation component in the shape of a pneumatic cylinder, as shown in Figure 123, is modelled as AutomationML Component.



Figure 123: Picture of the real pneumatic cylinder ADN-25-50-A-P-A that is modelled here as an AutomationML Component

This example is not intended to be complete but should give a clear overview how to model an AutomationML Component in several aspects.

Following contents of the cylinder are included within the AutomationML Component:

- Several typical datasheet attributes for identification, technical description and commercial use
- Picture, icon and pneumatic symbol of the component and a manufacturer icon
- Documentation of the component
- Geometry and kinematic models with linking to the appropriate COLLADA document
- Simplified logical behaviour model with linking to the appropriate AMLLogic document
- Mechanic connectors for rod, base and sensor slots of the component
- Pneumatic connectors of the component
- Several relations (internal links) between kinematic model, behaviour model and connectors

9.1.1 Container Package

The whole AutomationML Component description of this component is packed into one AutomationML Component Container (AMLX). Figure 124 gives an overview of the documents contained in the AMLX file. The root document the AML document (visualized closer in Figure 125) is the entry point and holds relations as references to all the other documents in the container.

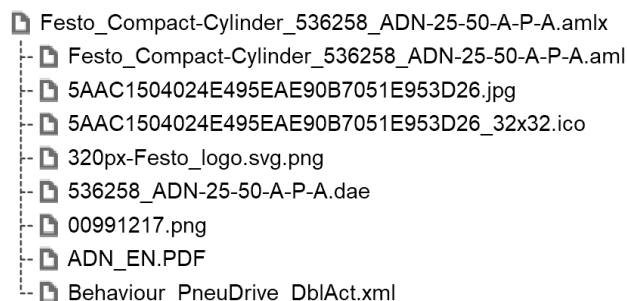


Figure 124: An unpacked tree view of the contained file of the AutomationML Component

Figure 125 shows the object tree of the component. All child elements of the object with the role class “AutomationComponent” form the component. The contents of the component are modelled as “InternalElements” and “InterfaceClasses” with the role classes and attributes as defined in this document and the related “AutomationComponentLibrary”. All rules are applied, and all references are set. “InternalLinks” model the inner relations between models to model and model to connectors and are visualized by blue lines here.



9.1.3 Connectors

As shown in Figure 126, the AutomationML Component has quite similar connectors as the automation component, the real pneumatic cylinder. There are two pneumatic connectors (InternalElements with role class “PneumaticConnector”) and there is a mechanic connector at the top of the rod and one on the base of the cylinder. There are three mechanic connectors at the sensor slots. Each of these mechanic connectors is represented by an InternalElement with the role class “MechanicConnector” (cp. Figure 125).

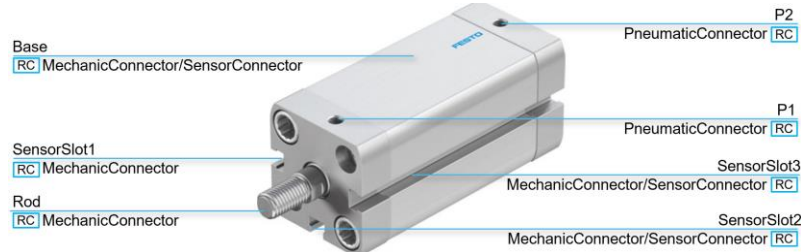


Figure 126: Overview of the modelled connectors of the AutomationML Component

9.1.4 Behaviour

Moreover, this component has a certain behaviour, if compressed air is conducted to only one of the pneumatic connectors the rod moves to a defined end position. This behaviour is modelled here to: Following the InternalLinks from the pneumatic connector to the InternalElement with the role class “BehaviourModel” in Figure 125, you can get a reference (attribute “refUri” from “ReferenceBehaviour” interface) to the AMLLogic document where this behaviour is described in the behaviour specific language AMLLogic. Figure 127 shows a visualization of this behaviour. Depending on the pressure state on the two inputs “PneumaticPort1” and “PneumaticPort2” the behaviour model returns a position value on the output “PositionRod”.

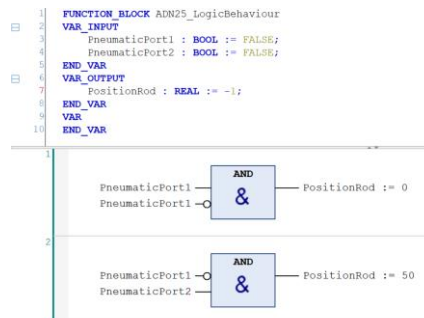


Figure 127: Contents of the AMLLogic behaviour model of the component

9.1.5 Kinematics and Geometry

In this example the behaviour model is also connected to the kinematic model of the component. The InternalLink on the behaviour models’ output “PositionRod” is connected to the interface “JointInterface” of an internalLink with the role class “COLLADAJointInterface”. It’s COLLADAInterface points to the joint definition into the COLLADA document that specifies the whole geometry and kinematics of the component. Inside the COLLADA document this joint is described as a prismatic joint between base and rod of the cylinder. By this relation between behaviour model and kinematic model it is clearly stated which part of the cylinder moves in which direction and position. Moreover, the COLLADA document describes attachment points published by InternalElements with the role class “COLLADAKinematicAttachment”. Figure 128 visualizes the kinematic model in the COLLADA document and the references used in the CAEX part.

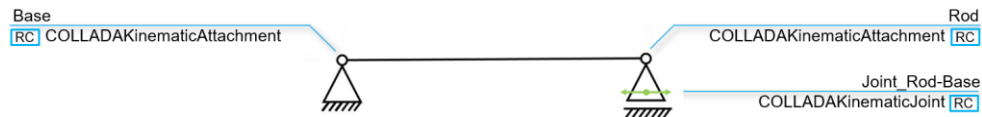


Figure 128: Visualization of the kinematic model of the component

In this example the kinematic attachments are connected by InternalLinks with the related mechanic connectors (see also Figure 126). Other AutomationML Components that are interconnected with their mechanic connectors, are to be seen as a fixed kinematic connection. E.g. a Gripper connected with its mechanic connector to the mechanic connector of the rod is meant to be moved with any movement of the rod.

9.1.6 Attributes

Attributes are attached to any object the AutomationML Component definition defines. Typical datasheet attributes for identification, technical description or commercial use is concentrated on the top-level InternalElement with the role class "AutomationComponent". Figure 129 shows an excerpt of the attributes of this component.

Name	Value	ID
IdentificationData		
Manufacturer	Festo SE & Co. KG	
ManufacturerURI	www.festo.com	
DeviceClass	pneumatic drive	
Model	ADN-25	
ProductCode	ADN-25-50-A-P-A	
OrderCode	ADN-25-50-A-P-A	
GeneralTechnicalData		
AmbientTemperature		
IPCode		
Material	Aluminium, Steel, TPU	
Weight	0,271	KG
Height	39,5	MM
Width	39,5	MM
Length	110,5	MM

Figure 129: Excerpt of attributes attached to the top-level element

9.1.7 Graphical Representation

Several graphical representations are part of this AutomationML Component, as shown in Figure 130. The InternalElement with the role class "ComponentPicture" shows a photographic view on this component. "ComponentIcon" shows a size reduced picture for the integration into tools. "PneumaticSymbol" is a typical pneumatic circuit element from engineering that shows the pneumatic function of the component in a human readable symbol. "ManufacturerIcon" is a logo from the manufacturer of this component and might also be used for a tool.

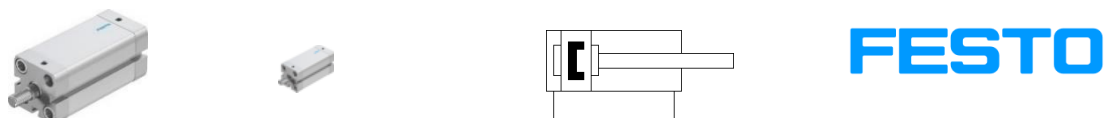


Figure 130: Content of "ComponentPicture", "ComponentIcon", "PneumaticSymbol", "ManufacturerIcon" (from left to right)

9.1.8 Documentation

The InternalElement with the role class "Documentation" points here to a user manual for this component. For the sake of simplicity only one document is attached here.

9.1.9 Summary

As a sum of all models, connectors, attributes, relations a comprehensive overall digital representation of the component is achieved. Not only useful datasheet attributes can be found but also picture and icons for the integration in a tool, that makes use of this AutomationML Component. Tools for virtual commissioning find interconnected models for geometry, kinematics and behaviour alongside with their relation to the connectors. Nevertheless even worthy information for classical engineering can be found.

9.2 Skill Example based on VDMA SOArc and VDI 2860 standards

9.2.1 Generic model of a skill based on VDMA SOArc and VDI standards

Different organizations have started standardizing different aspects of skills of automation components/systems. This example illustrates modelling the skill provided by a component/ system based on the standardization initiative by VDMA SOArc³ and VDI-2856⁴. VDMA SOArc defines a standard interface behaviour model, which abstract the logic behaviour of a skill, and offers an executable service interface to a component. By employing this standard on the components, the external consumers of the skills can execute skills in a unified manner. VDI-2856 defines a granularity (e.g. grip and move are lowest granularity skills) and naming conventions for skills. A generic model of skill of an automation component based on the standardized aspects of VDMA SOArc and VDI-2860 is depicted in.

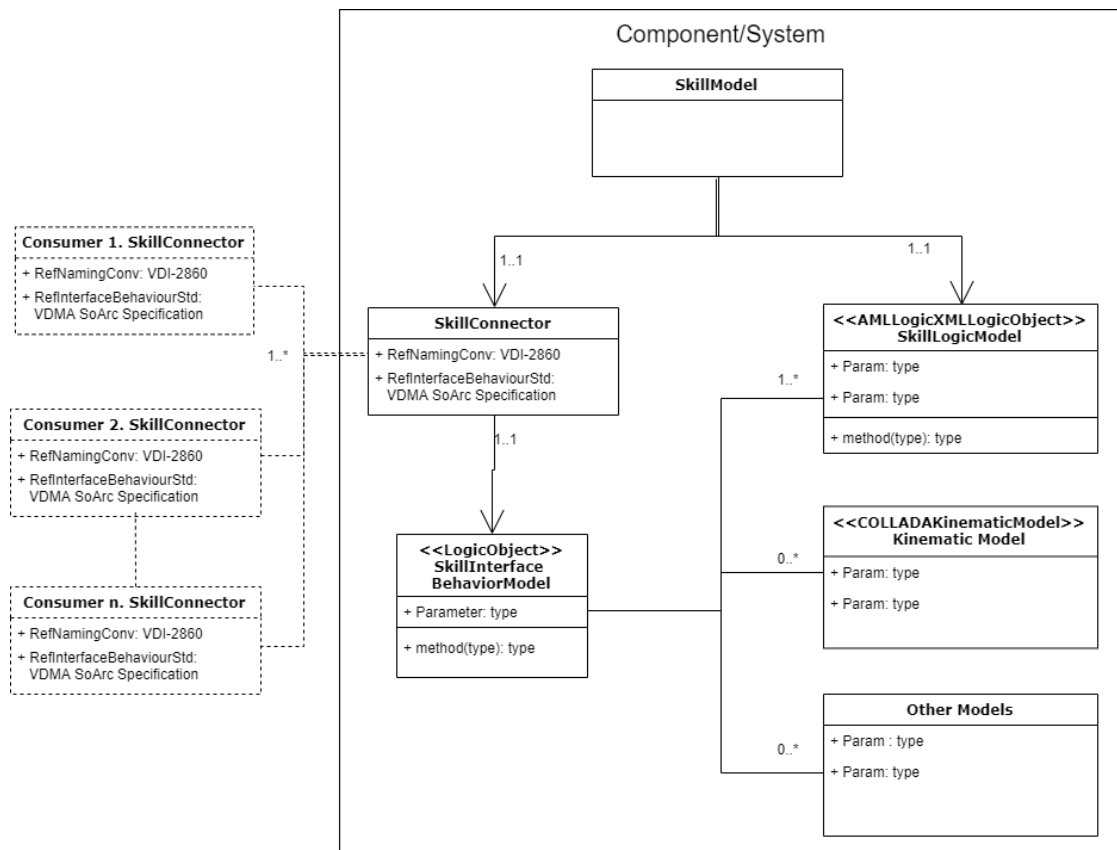


Figure 131: Class diagram showing different aspects of skill of an automation component

The figure illustrates the following attributes of skill of an automation component/system:

³ <https://ias.vdma.org/viewer/-/v2article/render/45533387>

⁴ <https://www.vdi.de/richtlinien/details/vdi-2860-handhabungsfunktionen-handhabungseinrichtungen-begriffe-definitionen-symbol>

- 1) The skill model has the potential to represent the functionality provided by an automation component/system (different domain specific models such as SkillLogicModel, KinematicModel, etc., implement the functional behaviour of a skill).
- 2) The model of a skill can be used as the connection point between different functional units that constitute an automated production system (the ComponentSkillConnector can be associated with different domain specific models implementing the functional behaviour of a skill).
- 3) Skill model is executable via a service interface (ComponentSkillConnector implemented based VDMA SOArc + VDI 2860 provides a standardized and executable service interface of the skill).
- 4) The skill provided by a component can be consumed by multiple consumers that require the skill (SkillConnectors of the consumers are associated to the SkillConnector of the component/system).

The detailed aspects of the SkillConnectors of consumers are out of scope of this document. The following section elaborates the details on modelling the skill provided by an automation component/system based on the standards VDMA SOArc and VDI 2860.

9.2.2 Modelling the provided skill of a component/system based on VDMA SOArc and VDI 2860 standards

This section assumes an automation component that provides a skill for external consumption. Examples are a Gripper that provide “Grip” and a linear drive that provide “Move”. The standardization aspects of VDMA SOArc and VDI-2860 modelled as a RoleClass is shown in Figure 132. The “VDMA SOArc+ VDI 2860” RoleClass extends the “SkillConnector” RoleClass provided in the AutomationMLComponentStdRCL. Additionally it uses the “SkillInterface” provided in the AutomationMLComponentBaseICL

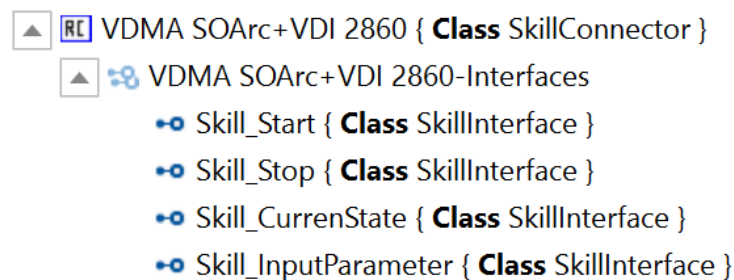


Figure 132: VDMA SOArc and VDI 2860 based SkillConnector RoleClass

For referring standards, the “VDMA SOArc+ VDI 2860” provides the attribute fields namely: “RefInterfaceBehaviourStd” and “RefNamingConvention”. Both these attributes provide the sub attributes described in

Table 85: Sub attributes for specifying the VDMA SOArc and VDI 2860 standards

Attribute	AttributeDataType	Description
Name	xs:string	The attribute “Name” shall be used to specify the used standard. The attribute is mandatory.
Version	xs:string	The attribute “Version” shall be used to specify a particular version of the standard. The attribute is mandatory.
URL	xs:string	The attribute “URL” shall be used to specify the URL where standard is defined. The attribute is optional.

Further, the RoleClass provides some of the interfaces specified in the VDMA SoArc. The “Skill_Start” is used to trigger a skill. The interface “Skill_Stop” stops the execution of a skill. The “Skill_CurrentState” provides the current state information of a skill such as “Executing” or “Idle”. “Skill_InputParameter” is used to specify the input parameter of a skill. When necessary the rest of the interfaces can be modelled in a similar manner.

The application of this RoleClass for modelling the skill provided by an automation component/system is depicted in Figure 133. The figure depicts the internal association of the SkillConnector and the SkillLogic Model. For example the “Skill_Start” and “Skill_Stop” is internally connected to the “Start/StopSignal” of a SkillLogicModel. The “OutputVariable” from the SkillLogicModel is connected to the “Skill_CurrentState”. The “Skill_InputParameter” is connected to the “InputParameter” of the SkillLogicModel. Further “Skill_Start” is connected to “Joint1”, which triggers the kinematic movement corresponding to the functionality provided by the skill.

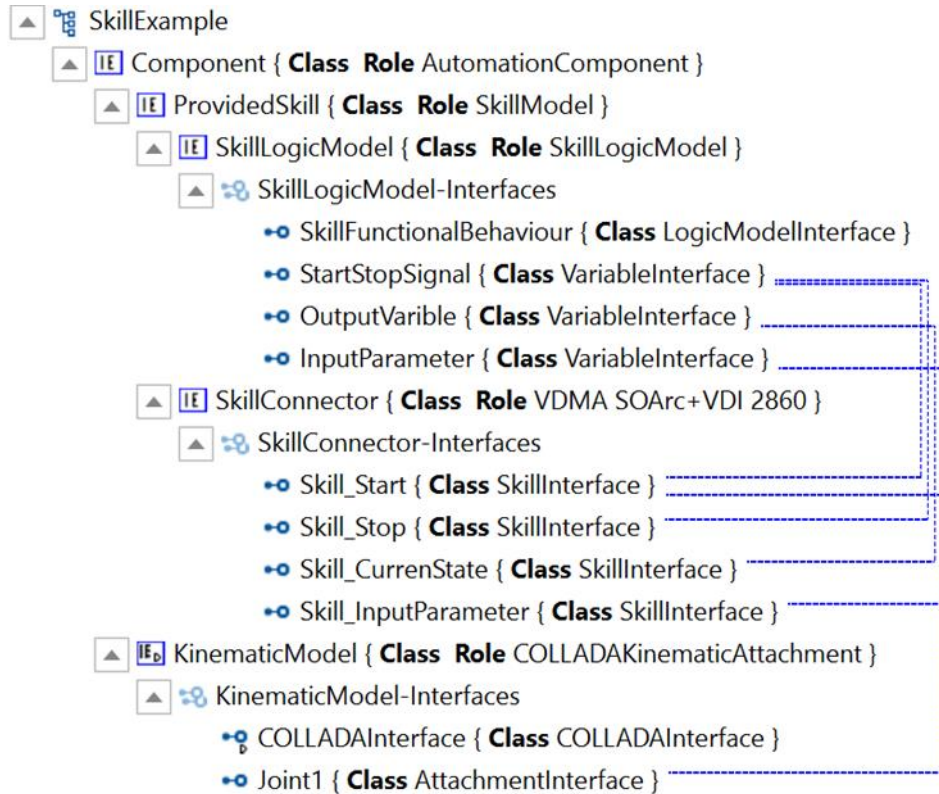


Figure 133: Automation component providing a skill

9.3 Connection of components

To demonstrate the connection of different AutomationML Components and the different model aspects of an AutomationML Component according to the specification of chapter 3.14 no own example is defined in this version of the document.

Aspects of the definitions can be found within the geometry and simulation practical examples in chapter 0 and 9.5.

9.4 Motor and Frequency Converter

This example shall show how a motor and a frequency converter can be modelled regarding virtual commissioning. Hereby also the modelling of behaviour and simulation based on standards (PLCopen XML) and user defined modules (FMU/FMI) shall be considered. Finally, the integration of these components with the PLC must be considered as well.

The model for simulation of a component needs:

- Inputs
- Outputs
- Parameters
- Constants
- Logic Behaviour (Link to PLCopen XML-Model)

Inputs list all the entries to the logic behavior element. The entry gates are used for attaching signals entering the logic behavior with links to PLC relevant data.

Outputs list all the outputs from the logic behavior element. The output gates are used for attaching signals exiting the logic behavior with links to PLC relevant data.

In case of compound components i.e. if a component is a sub-component we also have to define "InternalInputs" res. "External Inputs" to interact with the overlay components. These internal inputs/outputs shall not be visible in the overlay component.

Parameters list all the parameters in the logic behavior element. These are evaluated expressions used in other parameters and outputs. They can be seen as inputs in AutomationML.

Constants list all the constants in the logic behavior element. These are values used in parameters and outputs. They can be seen as inputs in AutomationML.

Logic Behaviour lists assignments to behaviour models. It is a link to a PLCopen XML-File and considers the use of Inputs and Outputs. It can also be a link to a user defined model.

9.4.1 Example Motor

The following example shall show this for a motor:

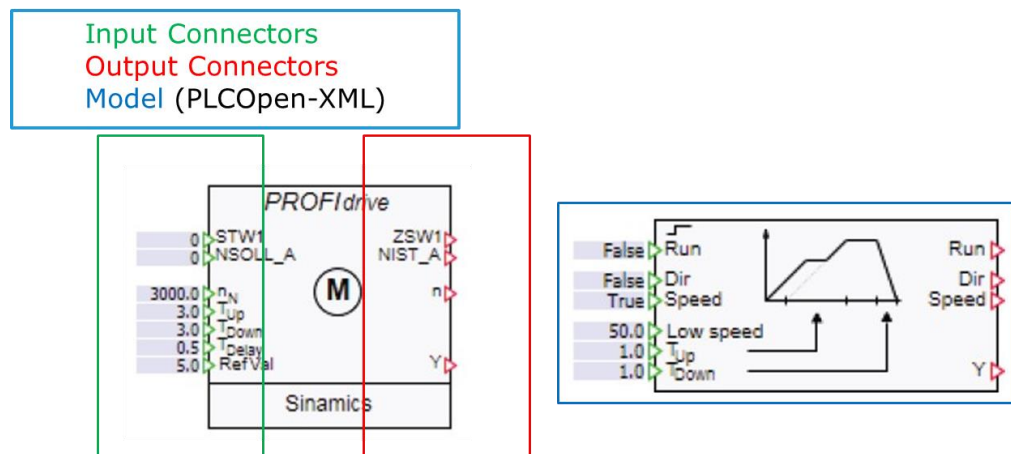


Figure 134: Example motor logic model

The Inputs are STW1, NSOLL_A. Parameters and Constants could be n_N , T_{up} , T_{down} , T_{delay} and RefVal dienen. The Outputs are in this example ZSW1, Nist_A, n and Y. The assigned logic for modelling the behaviour of a motor can be modelled as PLCopen XML or user defined model.

- ▲ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** Drive}
 - ▷ **IE** ElectricalConnectors {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▷ **IE** Simulation Models {**Class:** Model **Role:** Model}
 - SRC** AutomationProjectConfigurationDriveExtensionRoleClassLib/Drive

Figure 135: Example motor as internal element

The electrical connectors can be modelled as follows:

- ▲ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** Drive}
 - ▲ **IE** ElectricalConnectors {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▲ **IE** ElectricalConnectors-Interfaces
 - U1 {**Class:** ElectricInterface } ▷
 - V1 {**Class:** ElectricInterface } ▷
 - W1 {**Class:** ElectricInterface } ▷
 - PE {**Class:** ElectricInterface } ▷
 - SRC** AutomationMLComponentStandardRCL/ElectricConnector
 - ▷ **IE** Simulation Models {**Class:** Model **Role:** Model}
 - SRC** AutomationProjectConfigurationDriveExtensionRoleClassLib/Drive

Figure 136: Example motor definition of electrical connectors

The behavior can be modelled using the standard PLCOpen XML or a user defined model as well:

- ▲ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** Drive}
 - ▷ **IE** ElectricalConnectors {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▲ **IE** Simulation Models {**Class:** Model **Role:** Model}
 - ▲ **IE** StandardPLCOpenMotorModel {**Class:** PLCOpenXMLLogic **Role:** PLCOpenXMLLogic}
 - ▲ **IE** StandardPLCOpenMotorModel-Interfaces
 - Motor Behaviour {**Class:** BehaviourLogicModelInterface }
 - Run In {**Class:** VariableInterface } ▷
 - Dir In {**Class:** VariableInterface } ▷
 - Speed In {**Class:** VariableInterface } ▷
 - Low Speed In {**Class:** VariableInterface } ▷
 - Up In {**Class:** VariableInterface }
 - Down In {**Class:** VariableInterface }
 - Run Out {**Class:** VariableInterface }
 - Dir Out {**Class:** VariableInterface }
 - Speed Out {**Class:** VariableInterface }
 - Y Out {**Class:** VariableInterface }
 - SRC** AutomationMLComponentBaseRCL/LogicModel/PLCOpenXMLLogic
 - ▲ **IE** UserDefinedMotorModel {**Class:** SimulationModel **Role:** SimulationModel}
 - ▲ **IE** UserDefinedMotorModel-Interfaces
 - UserDefinedMotorBehaviour {**Class:** ExternalDataConnector }
 - UserDefined Run In {**Class:** SignalInterface } ▷
 - UserDefined Dir In {**Class:** SignalInterface } ▷
 - UserDefined Speed In {**Class:** SignalInterface } ▷
 - UserDefined Low Speed In {**Class:** SignalInterface } ▷
 - UserDefined Up In {**Class:** SignalInterface }
 - UserDefined Down In {**Class:** SignalInterface }
 - UserDefined Run Out {**Class:** SignalInterface }
 - UserDefined Dir Out {**Class:** SignalInterface }
 - UserDefined Speed Out {**Class:** SignalInterface }
 - UserDefined Y Out {**Class:** SignalInterface }
 - ExternalDataConnector {**Class:** ExternalDataConnector }
 - SRC** AutomationMLComponentStandardRCL/SimulationModel
 - SRC** AutomationMLComponentBaseRCL/Model
 - SRC** AutomationProjectConfigurationDriveExtensionRoleClassLib/Drive

Figure 137: Example motor integration of simulation model

9.4.2 Example Frequency Converter

A frequency converter can be modelled in the same way as a motor.

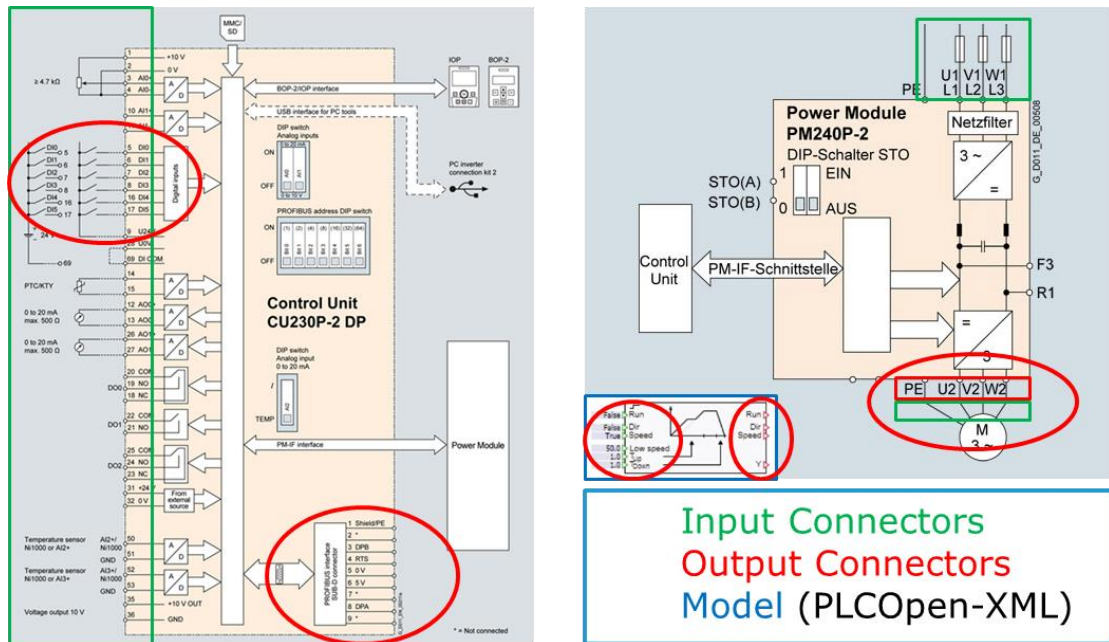


Figure 138: Overview example frequency converter

The control module of the frequency converter in this example has digital logical inputs for parametrization and is connected via Profibus to the PLC. The power module of the frequency converter is connected to the motor (U, V, W, PE). The control module is modelled in AutomationML via “SignalInterfaces”. The power module is modelled via “ElectricInterfaces”.

- ▲ **IE** SINAMICS G 120 {**Class:** Drive **Role:** Drive}
 - ▷ **IE** PROFINET interface {**Class:** Node **Role:** Node}
- ▲ **IE** Automation Input Interfaces SINAMICS {**Class:** LogicConnector **Role:** LogicConnector}
 - ▲ Automation Input Interfaces SINAMICS-Interfaces
 - SignalInterface DI 0 {**Class:** SignalInterface }
 - SignalInterface DI 1 {**Class:** SignalInterface }
 - SignalInterface DI 2 {**Class:** SignalInterface }
 - SignalInterface DI 3 {**Class:** SignalInterface }
 - SRC** AutomationMLComponentStandardRCL/LogicConnector
- ▲ **IE** Electric Output Interfaces SINAMICS {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▲ Electric Output Interfaces SINAMICS-Interfaces
 - U2 {**Class:** ElectricInterface }
 - V2 {**Class:** ElectricInterface }
 - W2 {**Class:** ElectricInterface }
 - PE {**Class:** ElectricInterface }
 - SRC** AutomationMLComponentStandardRCL/ElectricConnector
 - SRC** AutomationProjectConfigurationDriveExtensionRoleClassLib/Drive

Figure 139: Example frequency converter In-/Out interfaces

The connections between the frequency controller and the motor are modelled in Automation via Internal Links. Here the links for the power module:

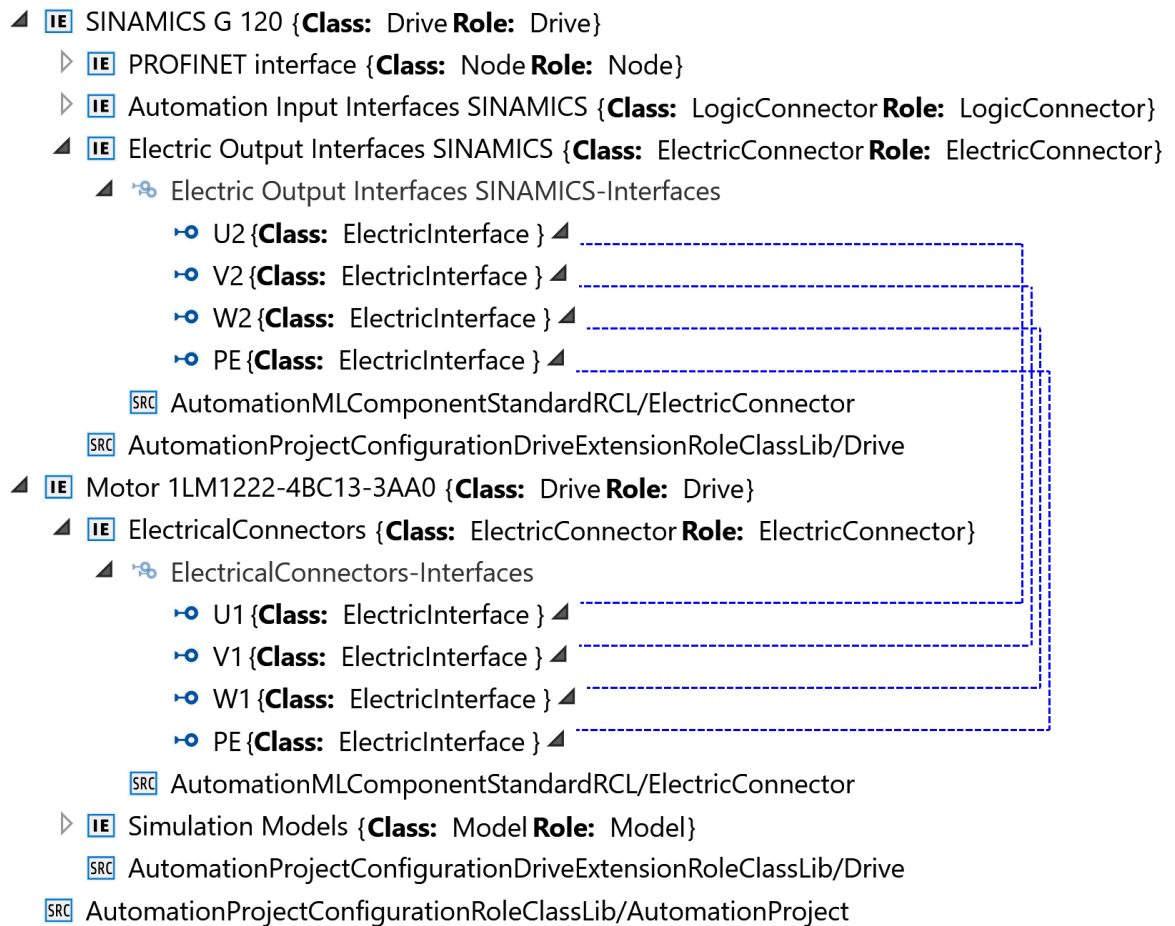


Figure 140: Example frequency converter connection to motor

9.4.3 Motor / Drive / PLC

Like the connections between frequency controller and motor the connections between the PLC and the frequency controller are modelled in Automation via Internal Links. In this example the PLC has digital outputs which shall be connected to the digital inputs of the frequency controller. Hereby the PLC is configured according AR APC:

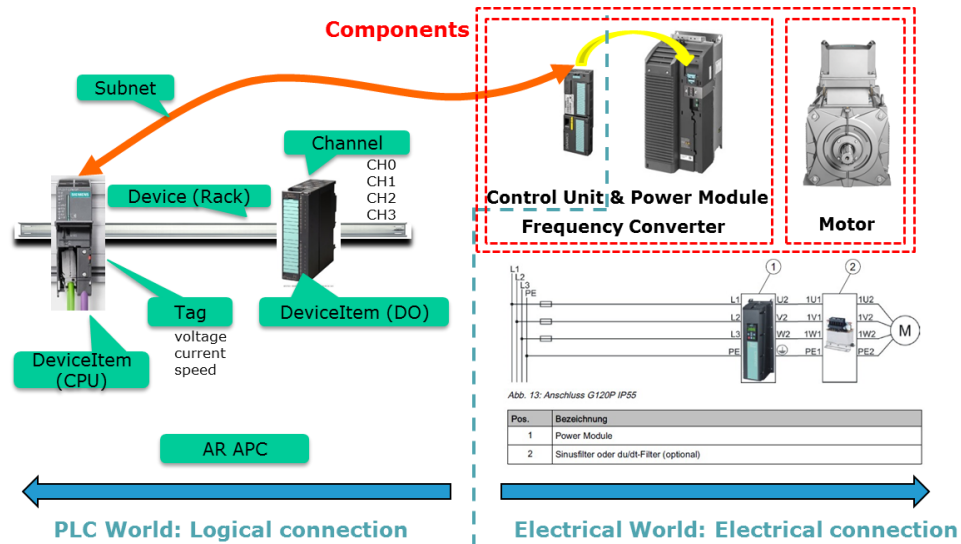


Figure 141: Example integration AR APC

So the complete example consists of a PLC station, a motor, a frequency controller and the Profibus:

- ▲ **IE** Project1 {**Class:** AutomationProject **Role:** AutomationProject}
 - ▷ **IE** PN/IE_1 {**Class:** Subnet **Role:** Subnet}
 - ▷ **IE** S71500/ET200MP station_1 {**Class:** Device **Role:** Device}
 - ▷ **IE** SINAMICS G 120 {**Class:** Drive **Role:** Drive}
 - ▷ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** Drive}
 - SRC** AutomationProjectConfigurationRoleClassLib/AutomationProject

Figure 142: Example integration AR APC project structure

Here the connection between the PLC and the frequency converter:

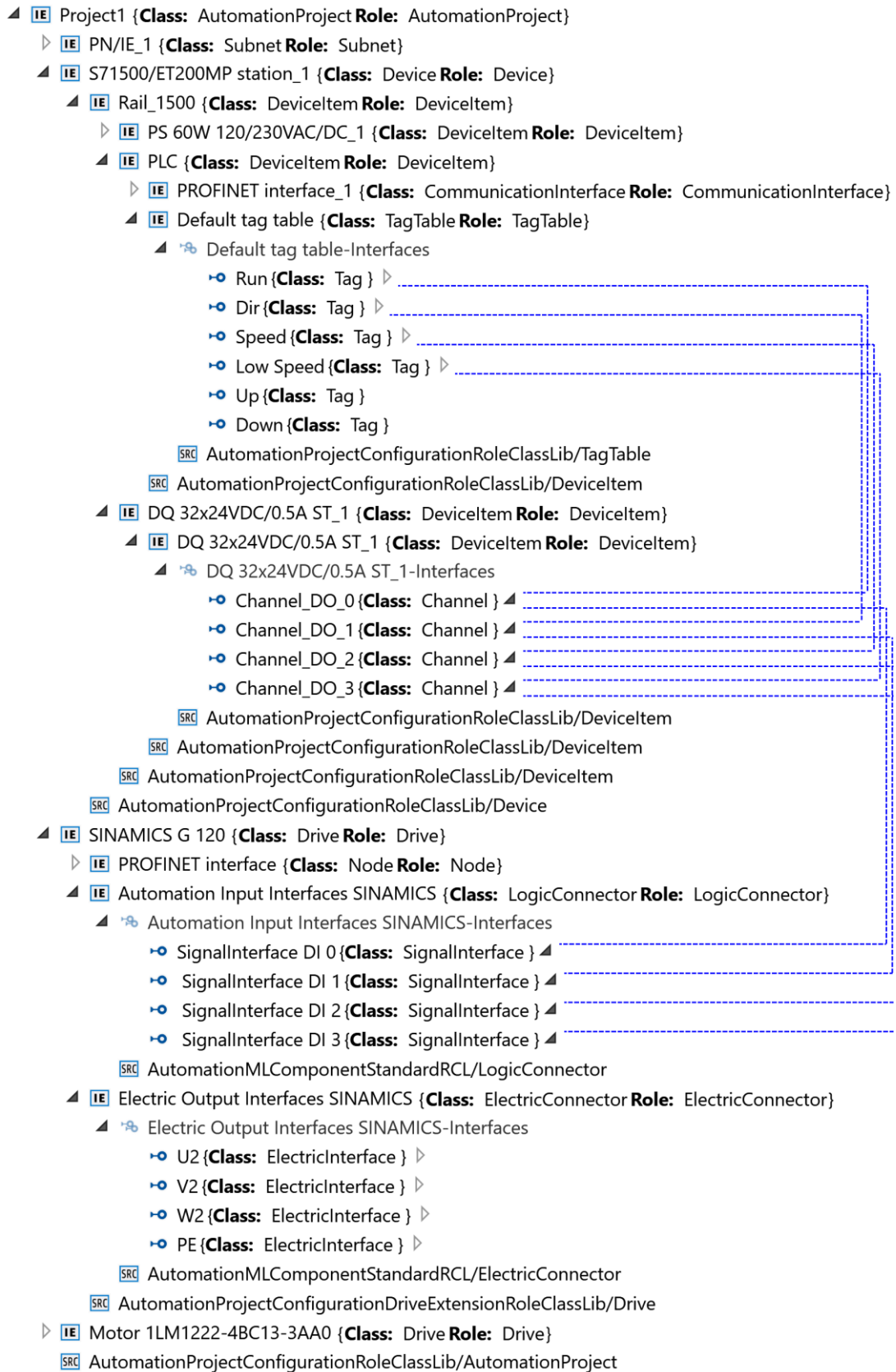


Figure 143: Example integration AR APC internal linking

9.5 User Defined Simulation Models

9.5.1 Referencing Functional Mockup Units According to the FMI Standard

The following section deals with the topic how to reference a Functional Mockup Unit (FMU) according to the FMI Standard. FMU's are often used for simulation and especially for the use case of virtual commissioning. The advantage of CoSimulation compiled FMU's is that for itself runnable simulation models - independent of the manufacturer - can handover and build together to a greater simulation system. This allows to simulate each kind of behavior independent of the complexity and enables an interdisciplinary engineering.

Many simulation tools bases on a signal flow structure. Defined inputs, parameters, outputs and executable code are characteristic for simulation models. If the system borders to other domains are declared it is lately just the question how to reference and instantiate FMU's into AutomationML.

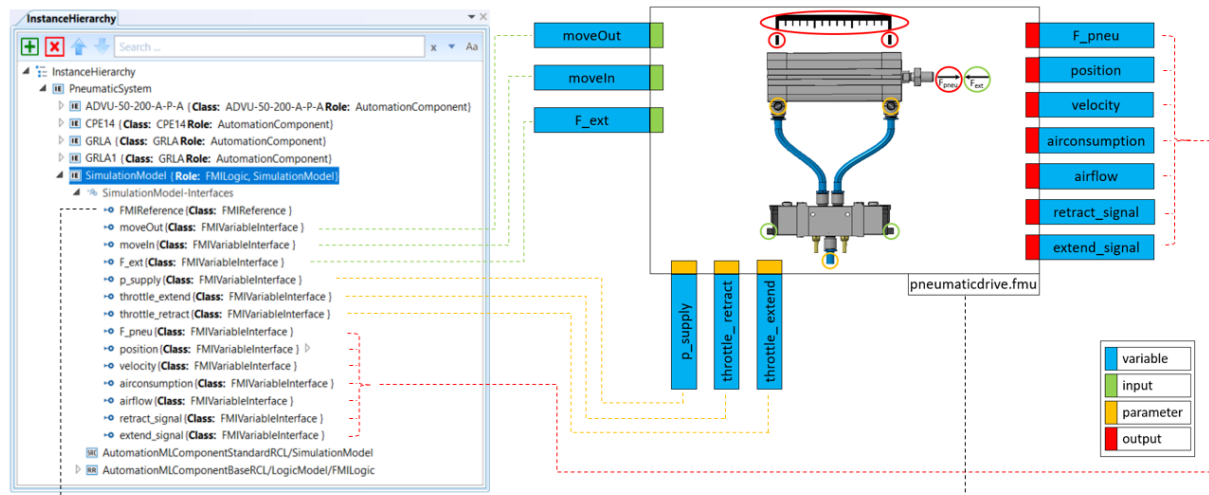


Figure 144: AutomationML Editor SimulationModel references an external FMU (left) – pneumatic drive.fmu (right)

Figure 144 shows the way how to reference between AutomationML and FMU's according the FMI standard. In principle the example of a PneumaticSystem could also be e.g. a robot or a electric drive. The scheme is always the same:

A InternalElement (SimulationModel) with the RoleClasses FMILogic and SimulationModel includes the path to the external FMU (pneumaticdrive.fmu) and variables. The InterfaceClasses FMIRreference and the FMIVariableInterface are the preferred Interfaces to connect the path of the external FMU and further information inside the modelDescription.xml which is part of every FMU.

The AutomationML variable attributes of the FMIVariableInterface refURI, Name and Causality described the FMU path, name and the property as input, parameter or output. With these AutomationML SimulationModel informations it is e.g. possible to connect kinematic joints of COLLADA files or PLCopenXML PLC programs to build up a interdisciplinary engineering file for different use cases.

9.5.2 SIMIT Simulation Models

This example shall show how a motor and a frequency converter can be modelled using vendor specific simulation models. The inputs and outputs are the same as in the PLCopen XML – example. But in difference to the connection to PLCopen XML the simulation code is provided and modelled within SIMIT. SIMIT can be used for a complete plant simulation including the simulation of signals, devices and plant responses. SIMIT provides an input and output simulator of test signals for an automation controller and allows testing and commissioning of automation software.

SIMIT also provides a lot of components for logical and arithmetic functions and for drives, sensors, connections and communication. Each component consists of an open XML based component interface (PORT) containing the various input and output connectors of the component and an encapsulated

content file (COMP) containing the simulation model for the behaviour of the component. This content is referenced by a SIMIT-UID.

Therefor it is sufficient to define a SIMIT shell containing an InterfaceClass derived from the LogicalEndpoint for the port and a RoleClass derived from the UserDefinedSimulationModel for embedding the encapsulated component:

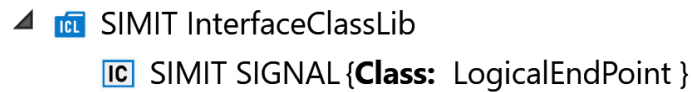


Figure 145: SIMIT InterfaceClassLibrary

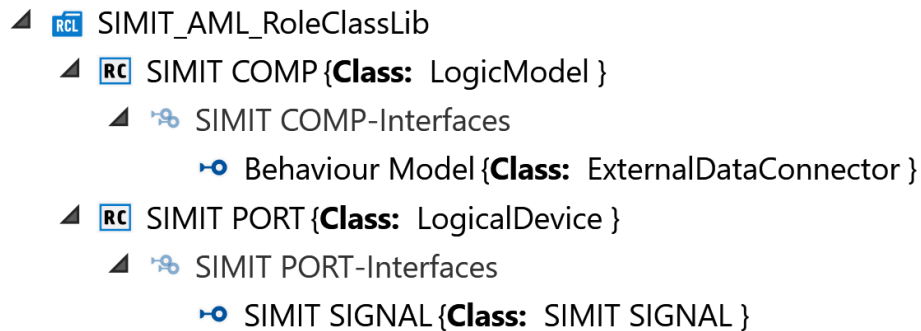


Figure 146: SIMIT RoleClassLibrary

Based on these two objects all SIMIT compnents can be mapped. The only difference in the motor frequency controller is the linkage to the SIMIT model instead of the PLCopen XML-model.

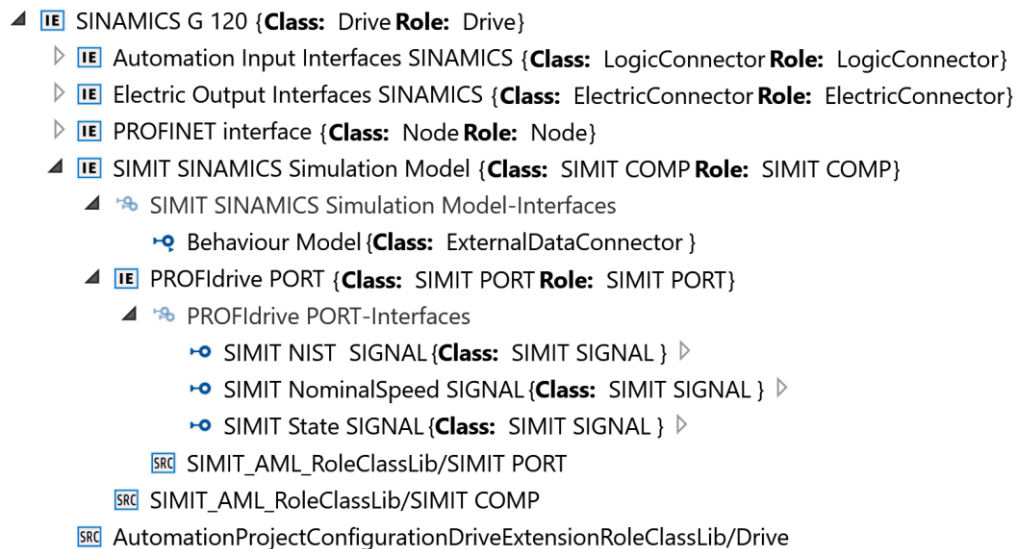


Figure 147: SIMIT Component Reference SINAMICS

- ▲ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** DeviceItem}
 - ▷ **IE** ElectricalConnectors {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▲ **IE** SIMIT Motor Simulation Model {**Class:** SIMIT COMP **Role:** SIMIT COMP}
 - ▲ **IE** SIMIT Motor Simulation Model-Interfaces
 - ▷ **IE** Behaviour Model {**Class:** ExternalDataConnector }
 - ▲ **IE** Start PORT {**Class:** SIMIT PORT **Role:** SIMIT PORT}
 - ▲ **IE** Start PORT-Interfaces
 - ▷ **IE** SIMIT SIGNAL {**Class:** SIMIT SIGNAL }
 - SRC** SIMIT_AML_RoleClassLib/SIMIT PORT
 - ▲ **IE** T_Up PORT {**Class:** SIMIT PORT **Role:** SIMIT PORT}
 - ▲ **IE** T_Up PORT-Interfaces
 - ▷ **IE** SIMIT SIGNAL {**Class:** SIMIT SIGNAL }
 - SRC** SIMIT_AML_RoleClassLib/SIMIT PORT
 - ▲ **IE** T_Down PORT {**Class:** SIMIT PORT **Role:** SIMIT PORT}
 - ▲ **IE** T_Down PORT-Interfaces
 - ▷ **IE** SIMIT SIGNAL {**Class:** SIMIT SIGNAL }
 - SRC** SIMIT_AML_RoleClassLib/SIMIT PORT
 - SRC** SIMIT_AML_RoleClassLib/SIMIT COMP
 - SRC** AutomationProjectConfigurationRoleClassLib/DeviceItem

Figure 148: SIMIT Component Motor Reference

The link to the SIMIT behaviour component is modelled according AutomationML as External Reference using the SIMIT UID as identifier within the refURI. The following figure shows an example for the SINAMICS frequency converter:

▲ IE SIMIT SINAMICS Simulation Model { Class: SIMIT COMP Role: SIMIT COMP}	Name	Value	DataType
▲ IE SIMIT SINAMICS Simulation Model-Interfaces	PortUID	..\DRIVES/PROFdrive/Sinamics interface.xml#d_000hsn_1yn7vor8	xs:anyURI
▷ IE Behaviour Model { Class: ExternalDataConnector }	PortName	PROFdrive	xs:string
▲ IE PROFdrive PORT { Class: SIMIT PORT Role: SIMIT PORT}	PortDirection	IN	xs:string
▲ IE PROFdrive PORT-Interfaces			
▷ IE SIMIT NIST SIGNAL { Class: SIMIT SIGNAL }			
▷ IE SIMIT NominalSpeed SIGNAL { Class: SIMIT SIGNAL }			
▷ IE SIMIT State SIGNAL { Class: SIMIT SIGNAL }			
SRC SIMIT_AML_RoleClassLib/SIMIT PORT			
SRC SIMIT_AML_RoleClassLib/SIMIT COMP			

Figure 149: SIMIT Component SINAMICS Reference

The input and output ports are modelled as SIMIT Signals derived from the SIMIT SIGNAL interface and linked with the tags of the PLC:

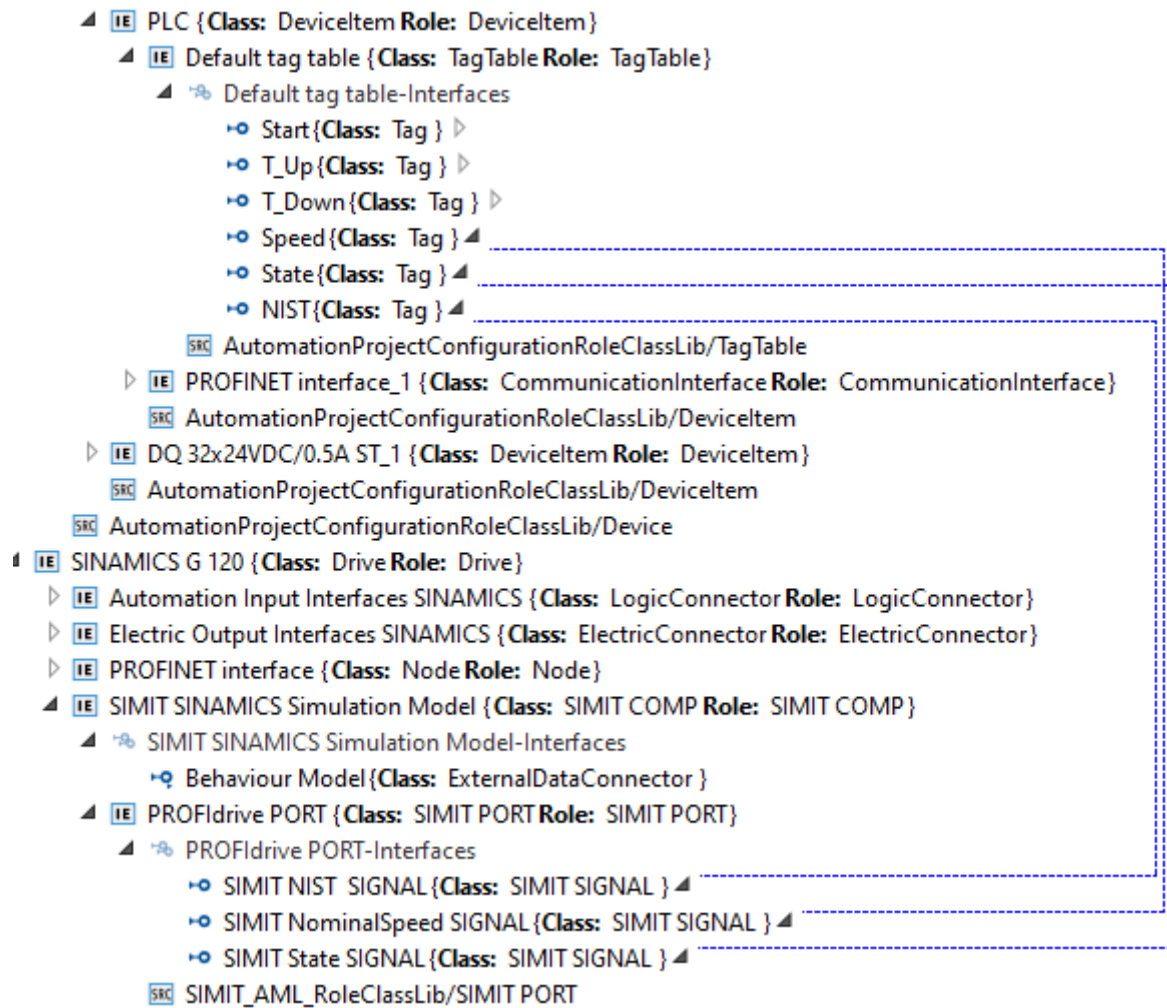


Figure 150: SIMIT Component SINAMICS Reference with PLC Signals

In the same way the motor component and the motor signals can be modelled:

	Name	Value	Data Type
<ul style="list-style-type: none"> IE SIMIT Motor Simulation Model {Class: SIMIT COMP Role: SIMIT COMP} <ul style="list-style-type: none"> SIMIT Motor Simulation Model-Interfaces <ul style="list-style-type: none"> Behaviour Model {Class: ExternalDataConnector} IE Start PORT {Class: SIMIT PORT Role: SIMIT PORT} <ul style="list-style-type: none"> Start PORT-Interfaces <ul style="list-style-type: none"> SIMIT SIGNAL {Class: SIMIT SIGNAL} 	refURI	..\DRIVES/Motor/Motor interface.xml#f_000hsn_4d99tg3u	xs:anyURI

Figure 151: SIMIT Component Motor Reference

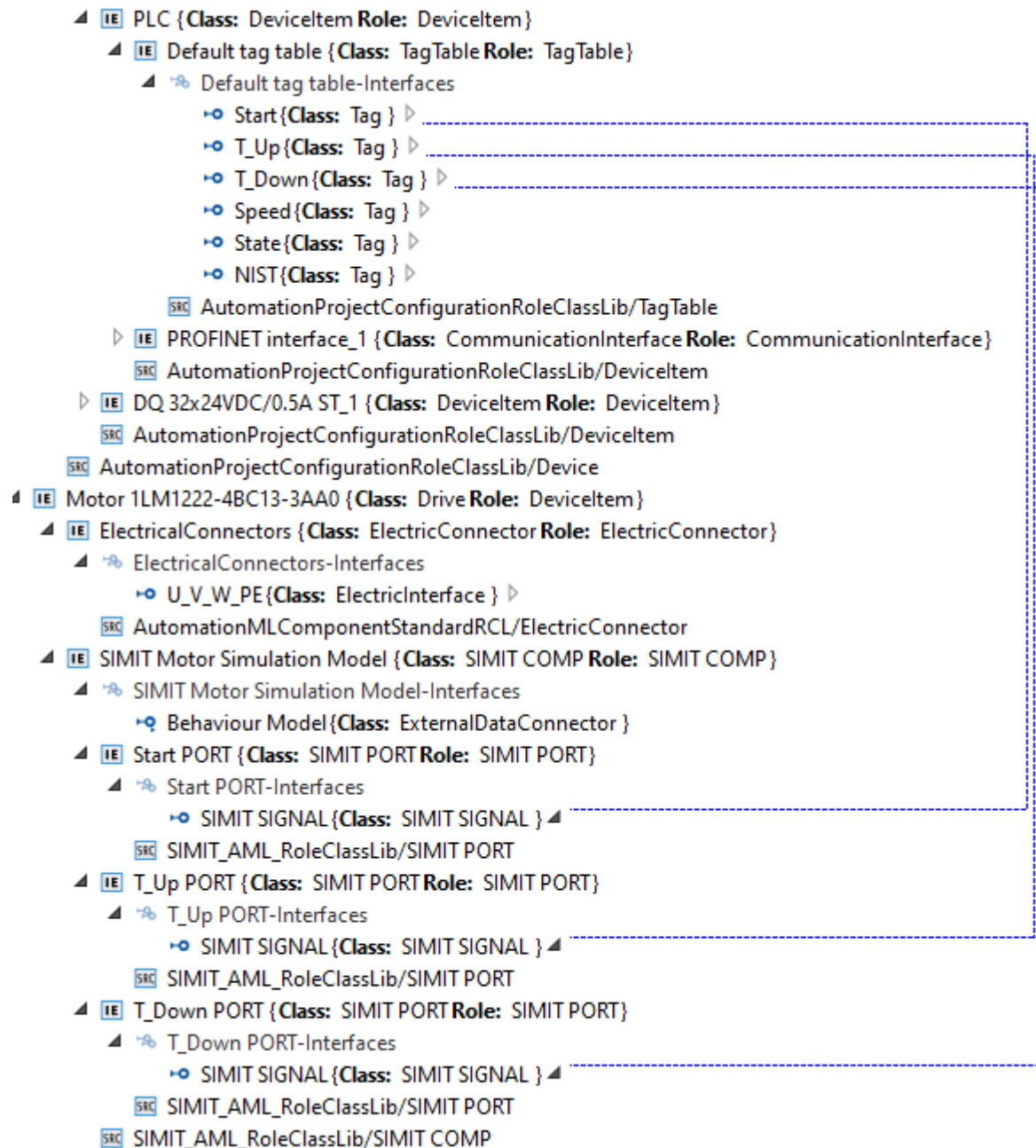


Figure 152: SIMIT Component Motor Reference with PLC Signals

The complete further configuration according AR APC remains unchanged. The connection to the SIMIT model is an extension to the already defined PLC configuration:

- ▲ **IE** Project1 {**Class:** AutomationProject **Role:** AutomationProject}
 - ▷ **IE** PN/IE_1 {**Class:** Subnet **Role:** Subnet}
 - ▷ **IE** S71500/ET200MP station_1 {**Class:** Device **Role:** Device}
 - ▲ **IE** SINAMICS G 120 {**Class:** Drive **Role:** Drive}
 - ▷ **IE** Automation Input Interfaces SINAMICS {**Class:** LogicConnector **Role:** LogicConnector}
 - ▷ **IE** Electric Output Interfaces SINAMICS {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▷ **IE** PROFINET interface {**Class:** Node **Role:** Node}
 - ▷ **IE** SIMIT SINAMICS Simulation Model {**Class:** SIMIT COMP **Role:** SIMIT COMP}
 - SRC** AutomationProjectConfigurationDriveExtensionRoleClassLib/Drive
 - ▲ **IE** Motor 1LM1222-4BC13-3AA0 {**Class:** Drive **Role:** DeviceItem}
 - ▷ **IE** ElectricalConnectors {**Class:** ElectricConnector **Role:** ElectricConnector}
 - ▷ **IE** SIMIT Motor Simulation Model {**Class:** SIMIT COMP **Role:** SIMIT COMP}
 - SRC** AutomationProjectConfigurationRoleClassLib/DeviceItem
 - SRC** AutomationProjectConfigurationRoleClassLib/AutomationProject

Figure 153: Complete PLC Configuration with SIMIT as extension

9.6 Additional Device Description

An example for the reference of additional device description can be found in [BPR-CLPAAML:2020].

9.7 Geometry and Kinematic Example

The following example shows the geometry and kinematic model of three different components, namely a motor, an adapter and a linear positioning axis. The components are from different vendors in order to show the interoperability of the here defined models. The examples shows models for each of the components separately including the geometry, kinematic models and their relations to the mechanical connectors as well as the models for the composite component making use of the three single components.

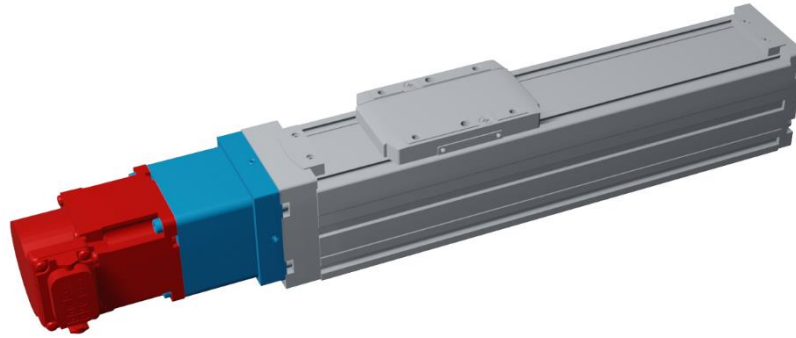


Figure 154: CAD Drawing of a motor (red), adapter (blue) and linear positioning axis (grey)

The single components are defined as SystemUnitClasses modelling them as types to be further used in the engineering process.

9.7.1 Motor

The system unit class shown in Figure 155 describing the motor HK-KT053W contains an InternalElement, which implements the role class “COLLADAGeometryModel”. It has an ExternalInterface of the class “COLLADAInterface” that references to the visual_scene in the COLLADA file. The value of its “refType” Attribute is “explicit”. The shaft of the motor has a relevance for other models and their interconnection. Therefore, a child InternalElement that implements the role class “COLLADAGeometryAttachment” is modelled. It has two ExternalInterfaces. One of the class “COLLADAInterface” which points to the visual_scene node and one of the class “AttachmentInterface” which can be used to define a geometrical coupling.

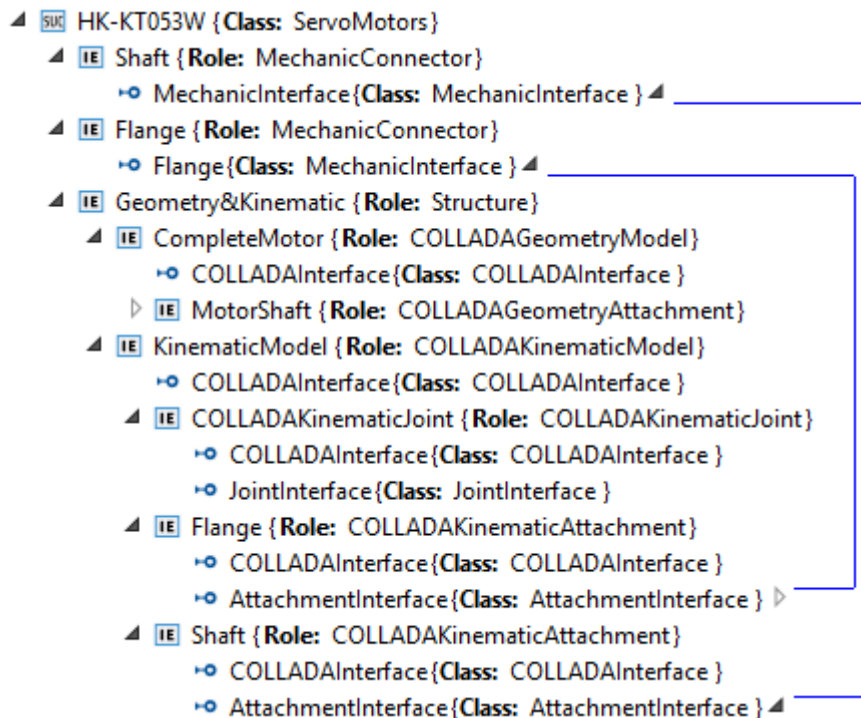


Figure 155: SystemUnitClass describing a motor with its geometry and kinematic model

For describing the kinematic model, the SUC contains an InternalElement, which implements the role class “COLLADAKinematicModel”. It has an ExternalInterface of the class “COLLADAInterface” which points to the kinematics_scene. The value of its refType Attribute is “explicit”.

A motor has usually one kinematic joint, which is a rotary joint rotates the shaft. It can be of interest for e.g. simulation where the angle value of this joint shall be connected to the variable of a simulation model. Therefore, an InternalElement, which implements the RoleClass “COLLADAKinematicJoint”, is created. It has an ExternalInterface of the class “COLLADAInterface” which references the kinematics_scene node. Additionally it has an ExternalInterface of the class “JointInterface”. This ExternalInterface can be used to link the joint to e.g. a simulation model.

A motor has two parts where kinematic coupling comes into play. One is the flange; the second one is the shaft. For both of them an own InternalElement, which implements the role class “COLLADAKinematicAttachment” is created. It defines is an ExternalInterface of the class “COLLADAInterface” which points to the kinematics_scene node. Additionally there is an “AttachmentInterface”. In order to define a kinematic coupling the “AttachmentInterface” is linked to the “MechanicInterface” of the corresponding “MechanicConnector”. An “InternalElement”, which implements the role class “MechanicConnector” and has an ExternalInterface of the class “MechanicInterface” exists for the shaft as well as for the flange. In later engineering steps, the “MechanicInterface” of one component will be linked to the one of another component. This will define the kinematic coupling between the “AttachmentInterfaces” of those components using the “MechanicConnector” as a proxy.

9.7.2 Adapter

Figure 154 shows that the motor and drive are connected with an adapter in between. Figure 156 shows how the aspects of geometry, kinematics and connectors of this adapter are modelled.

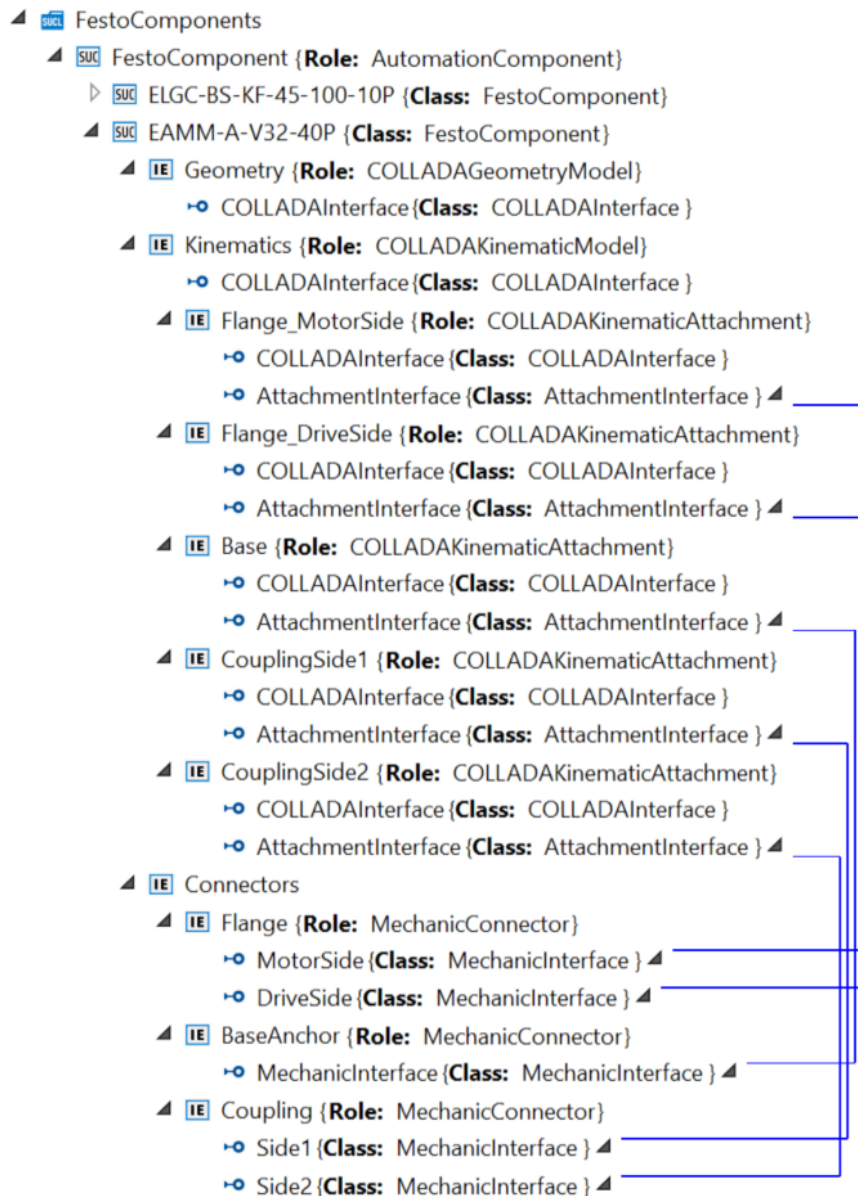


Figure 156: System unit class describing the motor-to-drive adapter with its geometry and kinematic mode

As the adapter EAMM-A-V32-40P (EAMM) comes from another vendor than the motor the structure and contents differs a bit to the motor. Nevertheless with the use of the standardized role classes “COLLADAGeometryModel” and “COLLADAKinematicModel” the sections for geometry and kinematics can be found easily.

The adapter mounts not only the housings of motor and drive with each other, but also the shafts of both with a coupling. Therefore a kinematic attachment is modelled for each of these four mounting points called Flange_MotorSide and Flange_Drive side for the mounting of the housings and CouplingSide1 and CouplingSide2 for the shafts. Each “COLLADAInterface” points to a frame node element inside the respective COLLADA file, with the refUri Attribute of the “COLLADAInterface”. Additionally the adapter might be fixed to a ground plate or be mounted on another drive. For this purpose the “COLLADAKinematicAttachment” called “Base” is modelled which virtually covers the whole adapter.

Finally these kinematics attachments are referenced with the real world representation of the connection points, the Connectors. As the adapter is fully mechanical part, all Connectors are “MechanicConnectors”. Figure 156 shows how the Connectors lead to the respective kinematic attachment of the kinematic Model.

9.7.3 Electromechanical Drive

The electromechanical drive translate the rotary movement from the motor and further transmitted by the adapter to a linear movement of a certain stroke. Figure 157 shows how the aspects of geometry, kinematics and connectors of this drive are modelled.



Figure 157: SystemUnitClass describing the drive with its geometry and kinematic model

The drive ELGC-BS-KF-45-100-100P (ELGC) is modelled in the same way the other components are modelled before. The section “Connectors” gives a good and short overview on the possible connection points. Besides the housing, here modelled as “BaseAnchor” there is the “Slide” which represents the moving linear slide for motion. Moreover there is the “Flange” and “Shaft” for connecting the motor via the adapter. Finally there are two “SensorSlots” for mounting slide position sensors to the drive. The blue lines show the InternalLinks to the respective kinematic attachments of the kinematic model that is modelled as parent element with the role class “COLLADAKinematicModel”. Each “COLLADAInterface” points to the respective section of the COLLADA file of this Automation Component

9.7.4 Interconnection of the components to a drive train

For the connection of the single components to a drive train only the connectors of each components needs to be connected, as in real world. Figure 5 shows how the connection of the single components to a drive train is modelled.

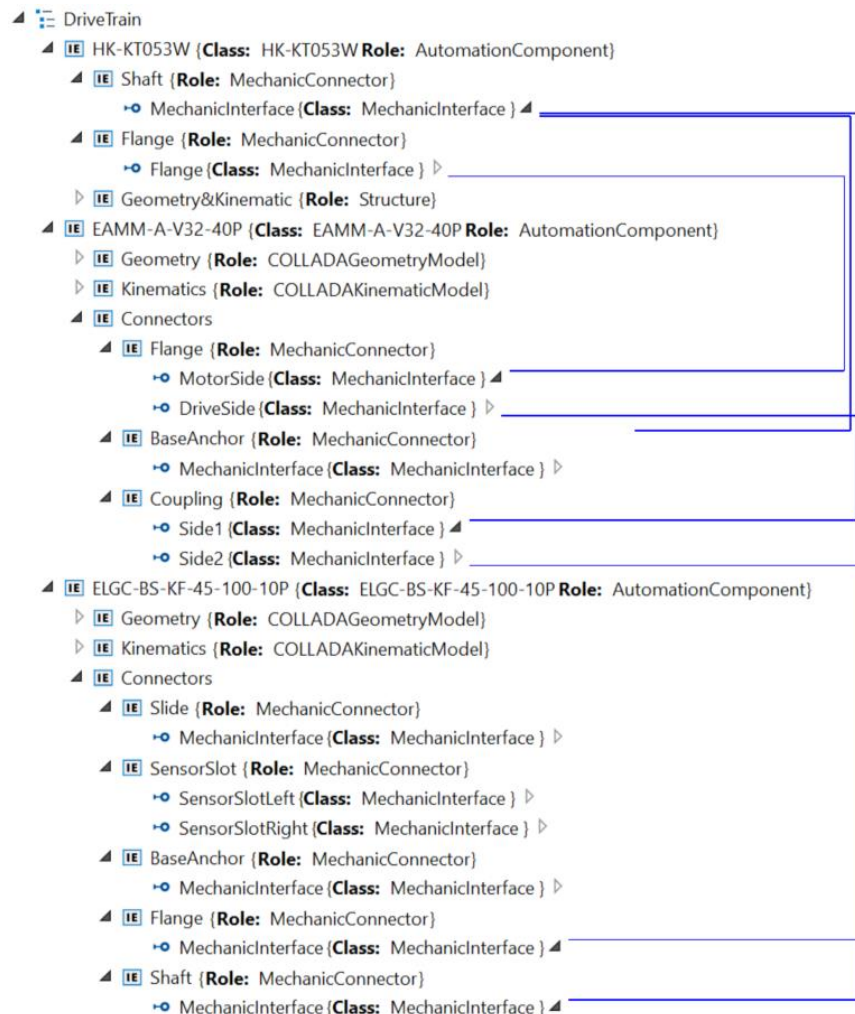


Figure 158: InstanceHierarchy of drive train build of single components

Figure 158 shows the three already described single components instantiated in the InstanceHierarchy. For the sake of clarity only the Connector section is expanded and the other, already described sections, are collapsed. Starting with the motor, the “MechanicInterface” of the Flange and Shaft are connected with InternalLinks with the motor side of the connectors of the adapter. The drive side of the adapter, Flange and Coupling, are connected with InternalLinks respectively with the “Flange” and “Shaft” of the electromechanical drive ELGC. The “MechanicConnector” of the drive “Slide” is not connected here, but can be used to be connected to an assembly that is mounted on the slide, e.g. a gripper. Also the “SensorSlots” are not used here.

By the connection of the MechanicConnector of the single components valuable information is stored in this model of drive train. Not only the configuration of this drive train is modelled but also the geometry and a full kinematic description is given here but the single models, geometry and kinematics, are put together to a full geometry and kinematics model of the drive train.

In detail, following the InternalLink path to the kinematic model of each component a tool can calculate the kinematics for the whole assembly and easily catch possible movements and limits. Furthermore, if these models shown here are enhanced by behaviour or simulation models such as FMILogic, AMLLogic or PLCOpenXMLLogic one can achieve virtual commissioning models exchanged in one AutomationML Component document.

9.8 Semantic of AutomationML Component Attributes

The example how to use different semantic systems within an AutomationML Component will be part of the next version of this whitepaper.

9.9 Modelling of a library of electrical M12 connector types

9.9.1 General

This chapter exemplarily illustrates the modelling of electrical interfaces by means of M12 connector types according to IEC61076 following the modelling provisions of the present Automation Component BPR related to CAEX 2.15.

9.9.2 M12 role class library

The M12 connector types are standardized in IEC61076 and are specified in multiple variants with different geometry (coding), number of pins, use cases and properties. A common M12 connector is A coded with 4 pins. The coding of M12 means a certain connector geometry, providing that only M12 connectors with the same coding fit together. Hence, multiple M12 variants with incompatible geometries exist.

An M12 connector contains sub-pins and hence has a nested characteristics: the connector contains interfaces. All M12 interfaces are modelled as role classes derived from the base class "ElectricConnector". Figure 159 shows a role class library IEC61076ConnectorRCL. The library demonstrates the modelling principles.

Remark: this model does not pursue a comprehensive modelling of all M12 connector types. Each of the M12 classes may model related attributes which are not part of this example.

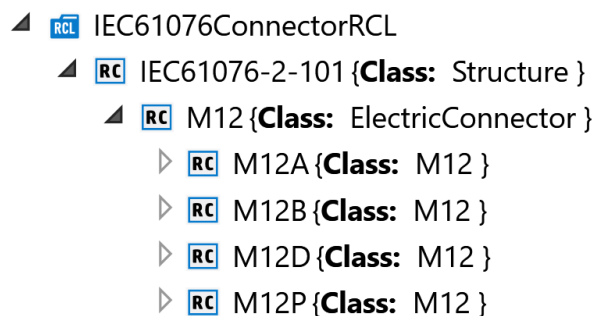


Figure 159: AML role class library modelling M12 connector types according to IEC61076-2.

Table 86: Description of connector types according to IEC61076-2

Class (of type)	Description
IEC61076ConnectorRCL (library)	AML Interface ClassLibrary modelling M12 connectors according to IEC61076
IEC61076-2-101 (Structure)	Abstract role class for M12 interfaces according to IEC61076-2-101
M12 (ElectricConnector)	base class for the M12 connector, derived by ElectricConnector according to the Automation Component model
M12A (M12)	Generic A coded M12 connector
M12B (M12)	Generic B coded M12 connector
M12D (M12)	Generic D coded M12 connector
M12P (M12)	Generic P coded M12 connector

Since M12 connectors contain pins with common properties, an interface class for a generic pin is designed. Figure 160 shows an CAEX InterfaceClassLibrary modelling a generic pin type derived from

the ElectricInterface. This class is the place to model required attributes related to pins, however they are not part of this example.



Figure 160: AML interface class library modelling a generic pin

Utilizing the InterfaceClass PinType, we can model the interfaces of all electric connector types. Figure 161 illustrates this by means of the type “M12A4Pin” which has 4 pins and a male/female variant.

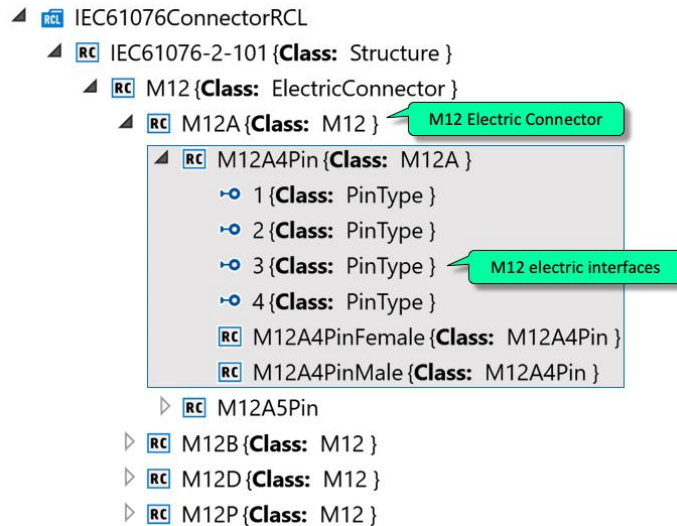


Figure 161: AML role class model of the M12 A coded with 4 pins and its male and female derivate

9.9.3 Example application role class library

The first step is the modelling of communication protocols as role library. This is shown in Figure 162.

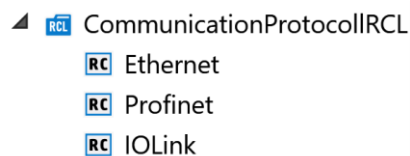


Figure 162: AML role class library modelling communication protocols

9.9.4 Application Example: Automation component with multiple electric connectors

Figure 163 shows an example product catalogue modelled as SystemUnitClasslibrary. Utilizing the M12 role class library shown in Figure 161, Figure 163 models an automation component “IOLink-MasterType123” with two M12 Ethernet connectors (A coded 5 Pins).

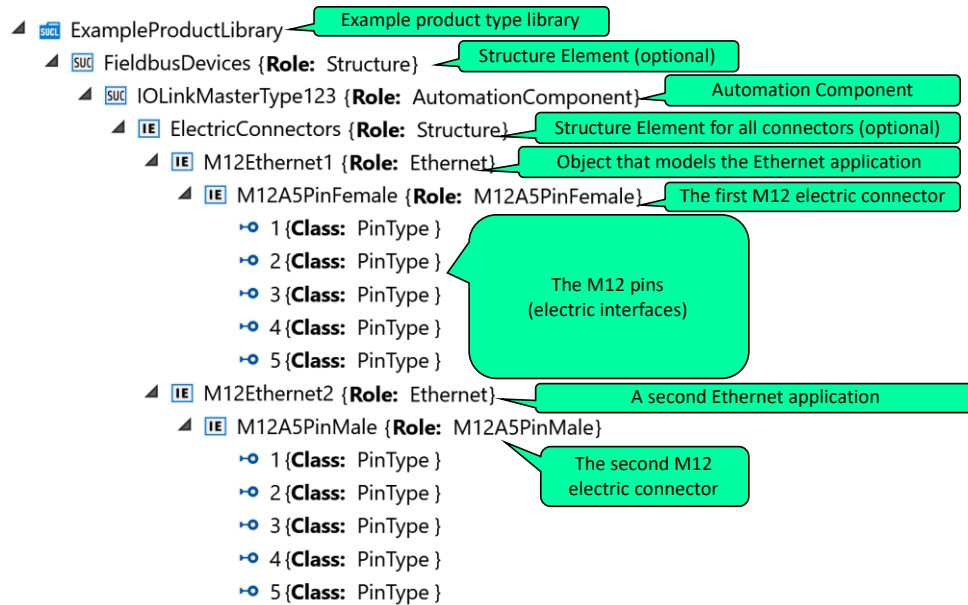


Figure 163: AML SystemUnitClass of an AutomationComponent with two M12 Ethernet connectors

9.9.5 Application Example: M12 to M12 cable

In this example, a cable with one M12 female and another M12 male connector is modelled and a modular modelling approach has been pursued. The modular modelling results in a CAEX System-UnitClass for a single wire, another SystemUnitClass for a cable with 4 wires, and a third System-UnitClass for the M12 to M12 cable which assembles all above classes.

Figure 164 shows the AutomationML model of a single wire with 2 interfaces. Related cable properties can be modelled here, they are not part of this example.

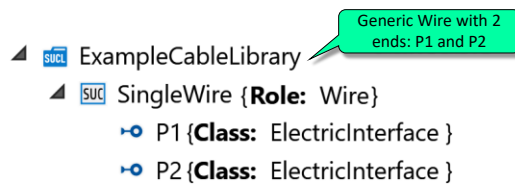


Figure 164: AML SystemUnitClass model of a single wire with two ends

Figure 165 assembles four of these single wires together in order to model a 4-wire-cable. Each single wire has own properties and inherits the interfaces.

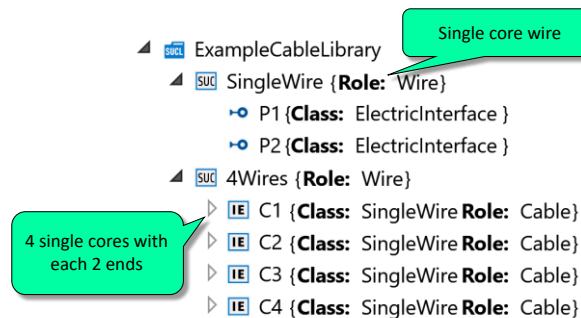
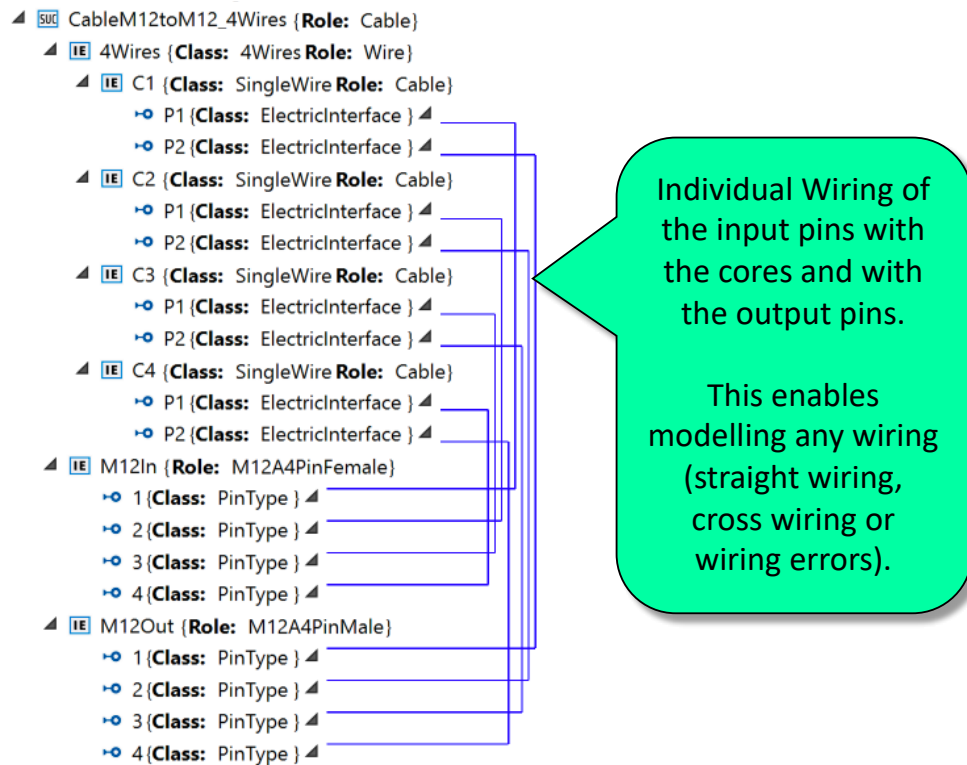


Figure 165: AML SystemUnitClass model of a cable with 4 wires

Figure 166 assembles the above classes in order to model M12-to-M12 cable (A-coded, 4 Pins). It consists of one 4-wire-cable, one M12 female interface and one M12 male interface. Furthermore, it models the wiring via CAEX InternalLinks.

The modelling of the internal connections is optional. However, it is useful for modelling different wiring types: straight wiring, cross wiring and even wiring errors or cable factories.



10 Variants of Automation Components in AutomationML

It is intended to specify a concept to describe variants for Automation Components in AutomationML. This will be included into Version 2 of this document.