VDMA 40010

ICS ...

Comments by 2018-06-19
Intended to replace
VDMA 40010:2018-06

OPC UA Robotics Companion Specification, part 1

Draft, Version 1.0, 2018-06-19

Continued on page 2

Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA)

VDMA
Price group ..

Contents

Contents **2**
Figure **6**
Tables **7**
1 Scope **9**
2 Normative references **11**
3 Terms, definitions and conventions **12**
3.1 Overview 12
3.2 OPC UA Robotics terms 12
3.2.1 Glossary 12
3.3 Abbreviations and symbols 13
3.4 Conventions used in this document 13
3.4.1 Conventions for Node descriptions 13
3.4.2 NodeIds and BrowseNames 15
3.4.3 Common Attributes 15
4 General information to OPC UA Robotics and OPC UA **19**
4.1 Introduction to OPC UA Robotics 19
4.2 Introduction to OPC Unified Architecture 19
4.2.1 What is OPC UA? 19
4.2.2 Basics of OPC UA 20
4.2.3 Information modelling in OPC UA 20
5 Use Cases **24**
6 OPC UA Robotics Information Model overview **26**
7 OPC UA ObjectTypes **28**
7.1 MotionDeviceSystemType ObjectType Definition 28
7.1.1 Overview 28
7.1.2 ObjectType definition 28
7.1.3 ObjectType description 29
7.2 MotionDeviceType ObjectType Definition 29
7.2.1 Overview 29
7.2.2 ObjectType definition 30
7.2.3 ObjectType description 31
7.3 AxisType ObjectType Definition 33
7.3.1 Overview 33
7.3.2 ObjectType definition 33
7.3.3 ObjectType description 34
7.4 PowerTrainType ObjectType Definition 35
7.4.1 Overview 35
7.4.2 ObjectType definition 35

7.4.3 ObjectType description 36
7.5 MotorType ObjectType Definition 37
7.5.1 Overview 37
7.5.2 ObjectType definition 37
7.5.3 ObjectType description 38
7.6 GearType ObjectType Definition 39
7.6.1 Overview 39
7.6.2 ObjectType definition 39
7.6.3 ObjectType description 39
7.7 SafetyStateType ObjectType Definition 40
7.7.1 Overview 40
7.7.2 ObjectType definition 41
7.7.3 ObjectType description 42
7.8 ControllerType ObjectType Definition 44
7.8.1 Overview 44
7.8.2 ObjectType definition 46
7.8.3 ObjectType description 46
7.9 ComponentType ObjectType Definition 48
7.9.1 Overview 48
7.9.2 ObjectType definition 48
7.9.3 ObjectType description 48
7.10 SoftwareType ObjectType Definition 49
7.10.1 Overview 49
7.10.2 ObjectType definition 49
7.10.3 ObjectType description 50
7.11 TaskControlType ObjectType Definition 50
7.11.1 Overview 50
7.11.2 ObjectType definition 51
7.11.3 ObjectType description 51
7.12 IdentificationType ObjectType Definition 52
7.12.1 Overview 52
7.12.2 ObjectType definition 53
7.12.3 ObjectType description 53
7.13 LoadType ObjectType Definition 54
7.13.1 Overview 54
7.13.2 ObjectType definition 55
7.13.3 ObjectType description 55
7.14 UserType ObjectType Definition 56
7.14.1 Overview 56
7.14.2 ObjectType definition 56

7.14.3 ObjectType description 56
8 OPC UA ReferenceTypes 57
8.1 General 57
8.2 Controls (IsControlledBy) Reference Type 57
8.3 Moves (IsMovedBy) Reference Type 58
8.4 Drives (IsDrivenBy) Reference Type 58
8.5 IsConnectedTo Reference Type 58
8.6 HasSafetyStates (SafetyStatesOf) Reference Type 59
9 OPC UA VariableTypes 60
9.1 EUItemType VariableType Definition 60
9.1.1 Overview 60
9.1.2 ObjectType definition 60
9.1.3 ObjectType description 60
9.2 VectorType VariableType 61
9.2.1 Overview 61
9.2.2 VectorType definition 62
9.2.3 VariableType description 62
9.3 PositionType VariableType 62
9.3.1 Overview 62
9.3.2 VariableType definition 63
9.3.3 VariableType description 63
9.4 FrameType VariableType 63
9.4.1 Overview 63
9.4.2 VariableType definition 64
9.4.3 VariableType description 64
10 OPC UA DataTypes 65
10.1 RationalNumber DataType 65
10.2 Vector DataType 65
10.3 Position DataType 65
10.4 Frame DataType 66
11 Profiles and Namespaces 67
11.1 Namespace Metadata 67
11.2 Conformance Units and Profiles 67
11.3 Server Facets 67
11.4 Client Facets 68
11.5 Handling of OPC UA Namespaces 68
A.1 Namespace and identifiers for <Title> Information Model 70
A.2 Profile URIs for <Title> Information Model 71
A.3 Profile URIs for <Title> Information Model 71
B.1 Examples of motion device systems, motion devices, axes and power trains 72
B.1.1 Example for motion device systems 72

B.1.2 Examples for motion devices and controllers in a motion device system 72
B.1.3 Examples for motion devices 72
B.1.4 Examples of combinations of motion devices in a motion device system 76
B.1.5 Axes and power trains 77
B.1.6 Virtual Axes 78
B.1.7 Examples for axes and power trains 78
B.1.8 Examples for the use of References regarding axes and power trains 78
C.1 Outlook and concepts 81
C.1.1 State machine 81
C.1.2 Geometry Reference 83
C.1.3 Geometrical representation of objects reference 86

Figure

Figure 1 – OPC UA standard definitions 17
Figure 2 – OPC UA and additional definitions 18
Figure 3 – The Scope of OPC UA within an Enterprise 20
Figure 4 – A Basic Object in an OPC UA Address Space 21
Figure 5 – The Relationship between Type Definitions and Instances 22
Figure 6 – Examples of References between Objects 23
Figure 7 – The OPC UA Information Model Notation 23
Figure 8 – Communication structure with OPC UA 25
Figure 9 – OPC UA Robotics describes the semantic self-description 25
Figure 10 – OPC UA Robotics top level view 26
Figure 11 – OPC UA Robotics overview 27
Figure 12 – Overview MotionDeviceSystemType 28
Figure 13 – Overview MotionDeviceType 30
Figure 14 – Overview AxisType 33
Figure 15 – Overview PowerTrainType 35
Figure 16 – Overview MotorType 37
Figure 17 – Overview GearType 39
Figure 18 – Overview SafetyStateType 41
Figure 19 – Overview ControllerType 45
Figure 20 – Overview ComponentType 48
Figure 21 – Overview SoftwareType 49
Figure 22 – Overview TaskControlType 51
Figure 23 – Overview IdentificationType 53
Figure 24 – Overview LoadType 54
Figure 25 – Overview UserType 56
Figure 26 – Reference Type Hierarchy 57
Figure 27 – Overview EUItemType 60
Figure 28 – Overview VectorType 61
Figure 29 – Overview PositionType 62
Figure 30 – Overview FrameType 63
Figure 31 – Cartesian manipulator 73
Figure 32 – Portal manipulator 73
Figure 33 – Stewart platform or Hexapod 74
Figure 34 – Delta robot 74
Figure 35 – Scara robot 75
Figure 36 – Articulated robot 75
Figure 37 – Schematic of a humanoid robot 76
Figure 38 – Motion device system 1 76
Figure 39 – Motion device system 2 77
Figure 40 – Axis and power train coupling 78
Figure 41 – *State machine* 82
Figure 42 – Example of layout description 84
Figure 43 – Abstracted server view of the geometry reference 85
Figure 44 – Simple frame chain 86
Figure 45 – Rotations 87
Figure 46 – Rotations of a frame 88

Tables

Table 1 – Terms and definitions 12
Table 2 – Abbreviations and definitions 13
Table 3 – Examples of DataTypes 14
Table 4 – Type Definition Table 15
Table 5 – Common Node Attributes 16
Table 6 – Common Object Attributes 16
Table 7 – Common Variable Attributes 16
Table 8 – Common VariableType Attributes 17
Table 9 – Common Method Attributes 17
Table 10 – Description of additional definitions 18
Table 11 – MotionDeviceSystemType Definition 28
Table 12 – TypeDefinition of MotionDevices of MotionDeviceSystemType 29
Table 13 – TypeDefinition of Controllers of MotionDeviceSystemType 29
Table 14 – TypeDefinition of SafetyStates of MotionDeviceSystemType 29
Table 15 – MotionDeviceType Definition 30
Table 16 – MotionDeviceClassEnumeration 31
Table 17 – ParameterSet of MotionDeviceType 31
Table 18 – TypeDefinition of Axes of MotionDeviceType 32
Table 19 –TypeDefinition of PowerTrains of MotionDeviceType 32
Table 20 – AxisType Definition 33
Table 21 – AxisMotionProfileEnumeration 34
Table 22 – ParameterSet of AxisType 34
Table 23 – PowerTrainType Definition 35
Table 24 – TypeDefinition of Motors of PowerTrainType 36
Table 25 – TypeDefinition of Gears of PowerTrainType 36
Table 26 – TypeDefinition of InternalDrives of PowerTrainType 36
Table 27 – MotorType Definition 37
Table 28 – ParameterSet of MotorType 38
Table 29 – GearType Definition 39
Table 30 – SafetyStateType Definition 41
Table 31 – TypeDefinition of EmergencyStopFunctions of SafetyStateType 42
Table 32 – ObjectType EmergencyStopFunctionType 42
Table 33 – TypeDefinition of ProtectiveStopFunctions of SafetyStateType 42
Table 34 – ObjectType ProtectiveStopFunctionType 43
Table 35 – Door Interlock Protective Stop Example 43
Table 36 – Teach Pendant Enabling Device Protective Stop Example 43
Table 37 – ParameterSet of SafetyStateType 44
Table 38 – OperationalModeEnumeration 44
Table 39 – ControllerType Definition 46
Table 40 – TypeDefinition of Components of ControllerType 46
Table 41 – TypeDefinition of Software of ControllerType 47
Table 42 – TypeDefinition of TaskControls of ControllerTyp 47
Table 43 – ParameterSet of ControllerType 47
Table 44 – ComponentType Definition 48
Table 45 – SoftwareType Definition 50
Table 46 – TaskControlType Definition 51
Table 47 – ParameterSet of TaskControlType 52
Table 48 – ExecutionModeEnumeration 52
Table 49 – IdentificationType Definition 53
Table 50 – LoadType Definition 55
Table 51 – LoadType possible degrees of modelling 55
Table 52 – UserType Definition 56
Table 53 – Controls Reference Definition 57
Table 54 – Controls Reference Definition 58

Table 55 – Drives Reference Definition 58
Table 56 – IsConnectedTo Reference Definition 58
Table 57 – HasSafetyStates Reference Definition 59
Table 58 – EUItemType Definition 60
Table 59 – VectorType Definition 62
Table 60 – PositionType Definition 63
Table 61 – FrameType Definition 64
Table 62 – RationalNumber Structure 65
Table 63 – RationalNumber Definition 65
Table 64 – Vector Structure 65
Table 65 – Vector Definition 65
Table 66 – Position Structure 65
Table 67 – Position Definition 66
Table 68 – Frame Structure 66
Table 69 – Frame TypeDefinition 66
Table 70 – NamespaceMetadata Object for this Specification 67
Table 71 – *Template Server Facet* Definition 68
Table 72 – *Template Client Facet* Definition 68
Table 73 – Namespaces used in a <title> Server 69
Table 74 – Namespaces used in this specification 69
Table 75 – Profile URIs 71

1 Scope

This specification was created by a joint working group of the OPC Foundation and VDMA. It defines an OPC UA Information Model for the representation of a complete motion device system as an interface for higher-level control and evaluation systems. A motion device system consists out of one or more motion devices, which can be any existing or future robot type (e.g. industrial robots, mobile robots), kinematics or manipulator as well as their control units and other peripheral components.

OPC Foundation

OPC is the interoperability standard for the secure and reliable exchange of data and information in the industrial automation space and in other industries. It is platform independent and ensures the seamless flow of information among devices from multiple vendors. The OPC Foundation is responsible for the development and maintenance of this standard.

OPC UA is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework. This multi-layered approach accomplishes the original design specification goals of:

– Platform independence: allows manufacturers independent exchange of information

– Scalable: from an embedded microcontroller to a cloud-based infrastructure

– Secure: encryption, authentication, authorization and auditing

– Expandable: ability to add new features including transports without affecting existing applications

– Comprehensive information modelling capabilities: for defining any model from simple to complex

VDMA Robotics Initiative

The VDMA is the biggest mechanical engineering industry association in Europe and represents over 3,200 mainly small and medium size member companies in the engineering industry, making it one of the largest and most important industrial associations in Europe. As part of the VDMA Robotics + Automation association, VDMA Robotics unites more than 75 members: companies offering robots, components of a robot, control units and motion device system integrations. The objective of this industry-driven platform is to support the robotics industry through a wide spectrum of activities and services such as standardization, statistics, marketing, public relations, trade fair policy, networking events and representation of interests.

Under the auspices of VDMA, a companion specification for robotics is developed by leading robot manufacturers and users within the "VDMA OPC Robotics Initiative". This Working Group has the status of an international joint working group with worldwide lead to develop a companion specification for robotics and is supported by the OPC Foundation. The aim is to create an information model with object types, which enables the modelling of robotic systems according to OPC UA as an interface for higher-level control and evaluation systems (plant control, MES, cloud). Not included are "application-related" interfaces, that can also be modelled via OPC UA. These interfaces are defined in further working groups for OPC UA Companion Specifications (e.g. EUROMAP 79, Integrated Assembly Solutions (e.g. gripper), IBV (Machine Vision)).

The VDMA Robotics Initiative is a working group within VDMA Robotics and was formed for the creation of this companion specification. The following members were actively involved in creating this document:

– ABB Automation GmbH

– Beckhoff Automation GmbH & Co. KG

– ENGEL AUSTRIA GmbH

– EPSON Deutschland GmbH

– fortiss An-Institut TU München

– Fraunhofer IGCV

– KEBA AG

- KraussMaffei Automation GmbH
- KUKA Deutschland GmbH
- Mitsubishi Electric Europe B.V.
- SIEMENS AG
- Unified Automation GmbH
- YASKAWA Europe GmbH

The following members provided further input for the working group:

- AUDI AG
- B+R automatizace, spol. s r.o.
- Daimler AG
- Microsoft Corporation
- Volkswagen AG

2 Normative references

The following documents, in whole or in part, are normatively 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.

– ISO 8373:2012 Robots and robotic devices — Vocabulary

– ISO 10218-1:2011 Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots

– EN 81346-2:2009 Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations – Part 2: Classification of objects and codes for classes (IEC 81346-2:2009)

– IEC 62541-3:2015 OPC UA Part 3: Address Space Model

– IEC 62541-4:2015 OPC UA Part 4: Services

– IEC 62541-5:2015 OPC UA Part 5: Information Model

– IEC 62541-8:2015 OPC UA Part 8: DataAccess

– IEC 62541-100:2015 OPC UA Part 100: Device Interface

3 Terms, definitions and conventions

For the purposes of this document, the following terms and definitions apply.

3.1 Overview

It is assumed that the reader of this document understands the basic concepts of OPC UA information modelling and the referenced documents. This specification will use these concepts to describe the Robotics Information Model.

Note that OPC UA terms and terms defined in this specification are written in *italics* in the specification.

3.2 OPC UA Robotics terms

3.2.1 Glossary

Table 1 – Terms and definitions

Term	Definition of Term
Asset management	The management of the maintenance of physical assets of an organization throughout each asset's lifecycle.
Automatic mode	Operating mode in which the robot control system operates in accordance with the task programme (ISO 10218).
Axis	The mechanical joint (ISO 8373). Joint is used as a synonym for axis.
Condition monitoring	Acquisition and processing of information and data that indicate the state of a machine over time (ISO 13372:2012).
Controller	Controlling unit of a one or more motion devices. A controller can be e.g. a specific control cabinet or a PLC.
Industrial robot	Automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications (ISO 10218).
Joint	See Axis definition.
Manual mode	Control state that allows for the direct control by an operator (ISO 10218).
Motion device	An industrial moving device, e.g. a industrial robot, positioner or mobile platform.
Motion device system	The whole system in which one or more motion devices are integrated, e.g. a robot cell.
Operator	Person designated to start, monitor and stop the intended operation of a robot or robot system (ISO 8373).
Teach pendant	Hand-held unit linked to the control system with which a robot can be programmed or moved (ISO 8373).
Power train	The composition of drives, motors and gears to provide the required torque to an axis.
Predictive maintenance	Maintenance performed as governed by condition monitoring programmes (ISO 13372:2012)
Preventive maintenance	Maintenance performed according to a fixed schedule, or according to a prescribed criterion, that detects or prevents degradation of a functional structure, system or component, in order to sustain or extend its useful life.
Protective stop	Type of interruption of operation that allows a cessation of motion for safeguarding purposes and which retains the programme logic to facilitate a restart (ISO 10218).

Safety states	Set of safety functions and states which are related to a motion device system, e.g. emergency stop, protective stop.
Software	Runtime software or firmware of the controller.
Task control	Execution engine that loads and runs task programs. Synomyms for a task control are a sequence control or a flow control.
Task program	Program running on the task control. Set of instructions for motion and auxiliary functions that define the specific intended task of the robot system (ISO 8373).
Tool centre point	Point defined for a given application with regards to the mechanical interface coordinate system (ISO 8373)
User level	Current assigned user role.
User roles	User roles consists of specific permissions to access different features within a software. Users can be assigned to roles.
Virtual axis	Virtual axis has no power trains directly assigned in this motion device.

In the Annex B.1 you can find examples of the described terms.

3.3 Abbreviations and symbols

Table 2 – Abbreviations and definitions

Abbreviation	Definition of Abbreviation
CPU	Central Processing Unit
DOF	Degrees of freedom
ERP	Enterprise Ressource Planning
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
MES	Manufacturing Execution System
PLC	Programmable logic controller
PMS	Preventive Maintenance System
TCP	Tool centre point
TCP/IP	Transmission Control Protocol/Internet Protocol
TCS	Tool Coordinate System
UPS	Uninterruptible Power Supply
URL	Uniform resource locator

3.4 Conventions used in this document

3.4.1 Conventions for Node descriptions

Node definitions are specified using tables (see Table 4).

Attributes are defined by providing the *Attribute* name and a value, or a description of the value.

References are defined by providing the *ReferenceType* name, the *BrowseName* of the *TargetNode* and its *NodeClass*.

– If the *TargetNode* is a component of the *Node* being defined in the table, the *Attributes* of the composed *Node* are defined in the same row of the table.

– The *DataType* is only specified for *Variables*; "[<number>]" indicates a single-dimensional array, for multi-dimensional arrays the expression is repeated for each dimension (e.g. [2][3] for a two-dimensional array). For all arrays the *ArrayDimensions* is set as identified by <number> values. If no <number> is set, the corresponding dimension is set to 0, indicating an unknown size. If no number is provided at all the *ArrayDimensions* can be omitted. If no brackets are provided, it identifies a scalar *DataType* and the *ValueRank* is set to the corresponding value (see OPC UA Part 3). In addition, *ArrayDimensions* is set to null or is omitted. If it can be Any or ScalarOrOneDimension, the value is put into "{<value>}", so either "{Any}" or "{ScalarOrOneDimension}" and the *ValueRank* is set to the corresponding value (see OPC UA Part 3) and the *ArrayDimensions* is set to null or is omitted. Examples are given in Table 3.

Table 3 – Examples of DataTypes

Notation	Data-Type	Value-Rank	Array-Dimensions	Description
Int32	Int32	-1	omitted or null	A scalar Int32.
Int32[]	Int32	1	omitted or {0}	Single-dimensional array of Int32 with an unknown size.
Int32[][]	Int32	2	omitted or {0,0}	Two-dimensional array of Int32 with unknown sizes for both dimensions.
Int32[3][]	Int32	2	{3,0}	Two-dimensional array of Int32 with a size of 3 for the first dimension and an unknown size for the second dimension.
Int32[5][3]	Int32	2	{5,3}	Two-dimensional array of Int32 with a size of 5 for the first dimension and a size of 3 for the second dimension.
Int32{Any}	Int32	-2	omitted or null	An Int32 where it is unknown if it is scalar or array with any number of dimensions.
Int32{ScalarOrOneDimension}	Int32	-3	omitted or null	An Int32 where it is either a single-dimensional array or a scalar.

– The TypeDefinition is specified for *Objects* and *Variables*.

– The TypeDefinition column specifies a symbolic name for a *NodeId*, i.e. the specified *Node* points with a *HasTypeDefinition Reference* to the corresponding *Node*.

– The *ModellingRule* of the referenced component is provided by specifying the symbolic name of the rule in the *ModellingRule* column. In the *AddressSpace*, the *Node* shall use a *HasModellingRule Reference* to point to the corresponding *ModellingRule Object*.

If the *NodeId* of a *DataType* is provided, the symbolic name of the *Node* representing the *DataType* shall be used.

Nodes of all other *NodeClasses* cannot be defined in the same table; therefore only the used *ReferenceType*, their *NodeClass* and their *BrowseName* are specified. A reference to another part of this document points to their definition.

Table 4 illustrates the table. If no components are provided, the DataType, TypeDefinition and ModellingRule columns may be omitted and only a Comment column is introduced to point to the *Node* definition.

Table 4 – Type Definition Table

Attribute	**Value**				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" will be used.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
ReferenceType name	*NodeClass* of the Target*Node*.	*BrowseName* of the target *Node*. If the *Reference* is to be instantiated by the server, then the value of the target Node's BrowseName is "--".	*DataType* of the referenced *Node*, only applicable for *Variables*.	*TypeDefinition* of the referenced *Node*, only applicable for *Variables* and *Objects*.	Referenced *ModellingRule* of the referenced *Object*.
NOTE Notes referencing footnotes of the table content.					

Components of Nodes can be complex that is containing components by themselves. The TypeDefinition, NodeClass, DataType and ModellingRule can be derived from the type definitions, and the symbolic name can be created as defined in chapter 3.4.3.1. Therefore, those containing components are not explicitly specified; they are implicitly specified by the type definitions.

3.4.2 NodeIds and BrowseNames

3.4.2.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. Annex A defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this specification is its *BrowseName*, or, when it is part of another *Node*, the *BrowseName* of the other *Node*, a ".", and the *BrowseName* of itself. In this case "part of" means that the whole has a *HasProperty* or *HasComponent Reference* to its part. Since all *Nodes* not being part of another *Node* have a unique name in this specification, the symbolic name is unique.

The namespace for all *NodeIds* defined in this specification is defined in Annex A. The namespace for this NamespaceIndex is *Server*-specific and depends on the position of the namespace URI in the server namespace table.

Note that this specification not only defines concrete *Nodes*, but also requires that some *Nodes* shall be generated, for example one for each *Session* running on the *Server*. The *NodeIds* of those *Nodes* are *Server*-specific, including the namespace. But the NamespaceIndex of those *Nodes* cannot be the NamespaceIndex used for the *Nodes* defined in this specification, because they are not defined by this specification but generated by the *Server*.

3.4.2.2 BrowseNames

The text part of the BrowseNames for all Nodes defined in this specification is specified in the tables defining the Nodes. The NamespaceIndex for all BrowseNames defined in this specification is defined in Annex A.

If the BrowseName is not defined by this specification, a namespace index prefix like '0:EngineeringUnits' or '2:DeviceRevision' is added to the BrowseName. This is typically necessary if a Property of another specification is overwritten or used in the OPC UA types defined in this specification. Table 74 provides a list of namespaces and their indexes as used in this specification.

3.4.3 Common Attributes

3.4.3.1 General

The *Attributes* of *Nodes*, their *DataTypes* and descriptions are defined in OPC UA Part 3. Attributes not marked as optional are mandatory and shall be provided by a *Server*. The following tables define if the *Attribute* value is defined by this specification or if it is server-specific.

For all *Nodes* specified in this specification, the *Attributes* named in Table 5 shall be set as specified in the table.

Table 5 – Common Node Attributes

Attribute	Value
DisplayName	The *DisplayName* is a *LocalizedText*. Each server shall provide the *DisplayName* identical to the *BrowseName* of the *Node* for the LocaleId “en”. Whether the server provides translated names for other LocaleIds is server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node.*
NodeId	The *NodeId* is described by *BrowseNames* as defined in 3.4.2.1.
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all non-server-specific *Attributes* to not writable. For example, the *Description Attribute* may be set to writable since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writable, because it is defined for each *Node* in this specification.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current Session can be provided. The value is server-specifc and depend on the *RolePermissions Attribute* (if provided) and the current *Session*.
AccessRestrictions	Optionally server-specific access restrictions can be provided.

3.4.3.2 Objects

For all *Objects* specified in this specification, the *Attributes* named in Table 6 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 6 – Common Object Attributes

Attribute	Value
EventNotifier	Whether the *Node* can be used to subscribe to *Events* or not is server-specific.

3.4.3.3 Variables

For all *Variables* specified in this specification, the *Attributes* named in Table 7 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 7 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a server-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is server-specific, for all other *Variables* defined in this specification, the access level shall allow reading; other settings are server-specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is server-specific. It is assumed that all *Variables* can be accessed by at least one user.
Value	For *Variables* used as *InstanceDeclarations,* the value is server-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *Variable*.
Historizing	The value for the *Historizing Attribute* is server-specific.
AccessLevelEx	If the *AccessLevelEx Attribute* is provided, it shall have the bits 8, 9, and 10 set to 0, meaning that read and write operations on an individual *Variable* are atomic, and arrays can be partly written.

3.4.3.4 VariableTypes

For all *VariableTypes* specified in this specification, the *Attributes* named in Table 8 be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 8 – Common VariableType Attributes

Attributes	Value
Value	Optionally a server-specific default value can be provided.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *VariableType*.

3.4.3.5 Methods

For all *Methods* specified in this specification, the *Attributes* named in Table 9 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 9 – Common Method Attributes

Attributes	Value
Executable	All *Methods* defined in this specification shall be executable (*Executable Attribute* set to "True"), unless it is defined differently in the *Method* definition.
UserExecutable	The value of the *UserExecutable Attribute* is server-specific. It is assumed that all *Methods* can be executed by at least one user.

3.4.3.6 Expanding conventions

For the following illustrations, the legend is as follows:

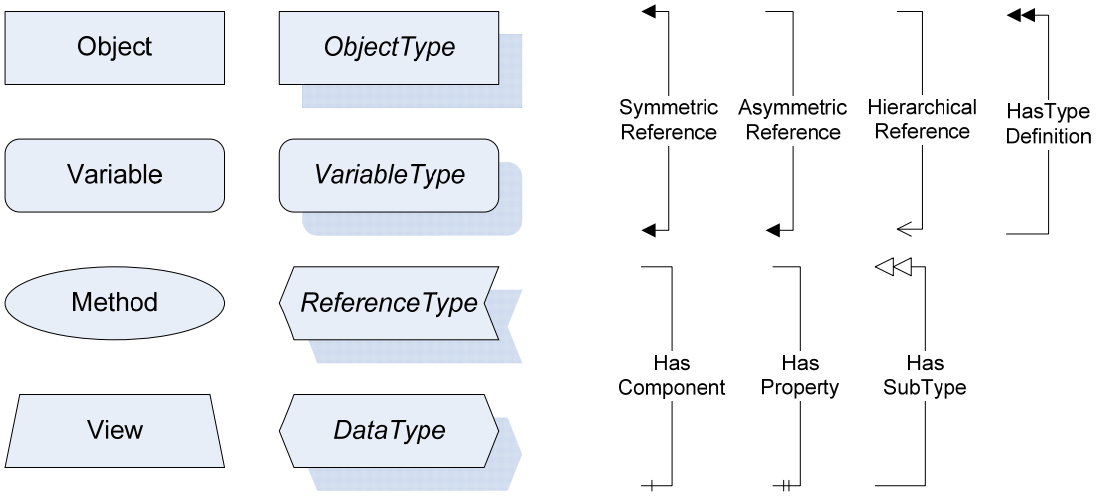

Figure 1 – OPC UA standard definitions

Additional definitions:

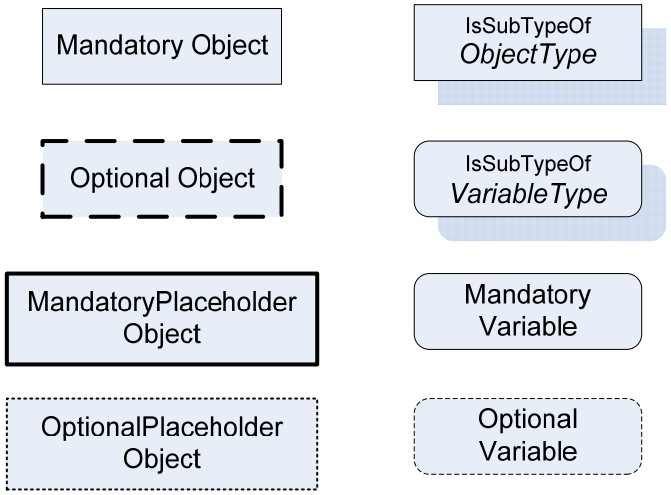

Figure 2 – OPC UA and additional definitions

Table 10 describes the additional definitions.

Table 10 – Description of additional definitions

Node element	Graphical representation	Definition of node element
Mandatory Object	Rectangular frame	A mandatory object with its type definition
Optional Object	Rectangular bold dashed frame	An optional object with its type definition
Mandatory Placeholder Object	Rectangular bold frame	A mandatory placeholder for objects with its type definition
Optional Placeholder Object	Rectangular dotted frame	An optional placeholder for objects with its type definition
ObjectType	Rectangular frame with shadow	An object type with its type definition
VariableType	Rounded rectangular frame with shadow	A variable type with its type definition
Mandatory Variable	Rectangular frame with rounded corners	A mandatory variable with its type definition
Optional Variable	Dotted rectangular frame with rounded corners	An optional variable with its type definition

3.4.3.7 Handling of not supported properties

In case of not supported *Properties* the following default shall be provided:

– *Properties* with *DataType* String: **empty string**

– *Properties* with *DataType* LocalizedText: **empty text field**

– *RevisionCounter Property*: **- 1**

4 General information to OPC UA Robotics and OPC UA

4.1 Introduction to OPC UA Robotics

The OPC UA Robotics specification describes an information model, which aims to cover all current and future robotic systems such as:

– Industrial robots

– Mobile robots

– Several control units

– Peripheral devices, which do not have their own OPC UA server

Part 1 provides information for asset management and condition monitoring. In future parts, the information model will be extended to cover more use cases.

The following functionalities are covered:

– Provision of asset configuration and runtime data of a running motion device system and its components e.g. manipulators, axes, motors, controllers and software

Following functions are not included and might be covered in future parts:

– A messaging mechanism covered by events and alarms to provide conditions

– A state machine to inform about the status of task controls and to interact via methods

  An outlook is descriped in C.1.1 StateMachine.

– The possibility for the operator to store customer specific information inside the motion device system e.g. location, cost center, ERP data, ...

4.2 Introduction to OPC Unified Architecture

4.2.1 What is OPC UA?

OPC UA is an open and royalty free set of standards designed as a universal communication protocol. While there are numerous communication solutions available, OPC UA has key advantages:

– A state of art security model (see OPC UA Part 2).

– A fault tolerant communication protocol.

– An information modelling framework that allows application developers to represent their data in a way that makes sense to them.

OPC UA has a broad scope which delivers for economies of scale for application developers. This means that a larger number of high quality applications at a reasonable cost are available.

The OPC UA model is scalable from small devices to ERP systems. OPC UA *Servers* process information locally and then provide that data in a consistent format to any application requesting data - ERP, MES, PMS, Maintenance Systems, HMI, Smartphone or a standard Browser, for examples. For a more complete overview see OPC UA Part 1.

4.2.2 Basics of OPC UA

As an open standard, OPC UA is based on standard internet technologies, like TCP/IP, HTTP, Web Sockets.

As an extensible standard, OPC UA provides a set of *Services* (see OPC UA Part 4) and a basic information model framework. This framework provides an easy manner for creating and exposing vendor defined information in a standard way. More importantly all OPC UA *Clients* are expected to be able to discover and use vendor-defined information. This means OPC UA users can benefit from the economies of scale that come with generic visualization and historian applications. This specification is an example of an OPC UA *Information Model* designed to meet the needs of developers and users.

OPC UA *Clients* can be any consumer of data from another device on the network to browser based thin clients and ERP systems. The full scope of OPC UA applications is shown in Figure 3.

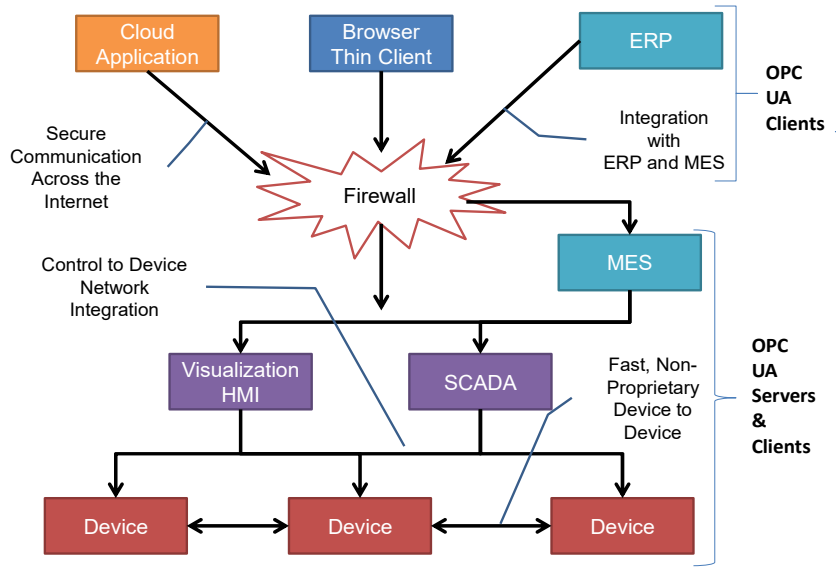

Figure 3 – The Scope of OPC UA within an Enterprise

OPC UA provides a robust and reliable communication infrastructure having mechanisms for handling lost messages, failover, heartbeat, etc. With its binary encoded data, it offers a high-performing data exchange solution. Security is built into OPC UA as security requirements become more and more important especially since environments are connected to the office network or the internet and attackers are starting to focus on automation systems.

4.2.3 Information modelling in OPC UA

4.2.3.1 Concepts

OPC UA provides a framework that can be used to represent complex information as *Objects* in an *AddressSpace* which can be accessed with standard services. These *Objects* consist of *Nodes* connected by *References*. Different classes of *Nodes* convey different semantics. For example, a *Variable Node* represents a value that can be read or written. The *Variable Node* has an associated *DataType* that can define the actual value, such as a string, float, structure etc. It can also describe the *Variable* value as a variant. A *Method Node* represents a function that can be called. Every *Node* has a number of *Attributes* including a unique identifier called a *NodeId* and non-localized name called as *BrowseName*. An *Object* representing a 'Reservation' is shown in Figure 4.

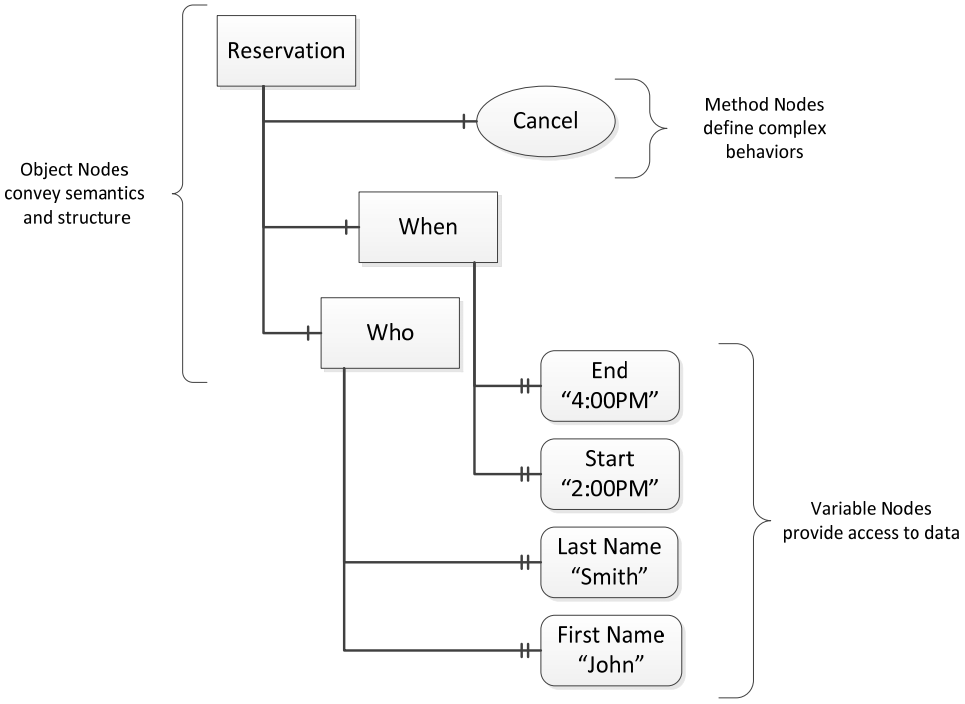

Figure 4 – A Basic Object in an OPC UA Address Space

Object and *Variable Nodes* represent instances and they always reference a *TypeDefinition* (*ObjectType* or *VariableType*) *Node* which describes their semantics and structure. Figure 5 illustrates the relationship between an instance and its *TypeDefinition*.

The type *Nodes* are templates that define all of the children that can be present in an instance of the type. In the example in Figure 5 the PersonType *ObjectType* defines two children: First Name and Last Name. All instances of PersonType are expected to have the same children with the same *BrowseNames*. Within a type the *BrowseNames* uniquely identify the children. This means *Client* applications can be designed to search for children based on the *BrowseNames* from the type instead of *NodeIds*. This eliminates the need for manual reconfiguration of systems if a *Client* uses types that multiple *Servers* implement.

OPC UA also supports the concept of sub-typing. This allows a modeller to take an existing type and extend it. There are rules regarding sub-typing defined in OPC UA Part 3, but in general they allow the extension of a given type or the restriction of a *DataType*. For example, the modeller may decide that the existing *ObjectType* in some cases needs an additional *Variable*. The modeller can create a subtype of the *ObjectType* and add the *Variable*. A *Client* that is expecting the parent type can treat the new type as if it was of the parent type. Regarding *DataTypes*, subtypes can only restrict. If a *Variable* is defined to have a numeric value, a sub type could restrict it to a float.

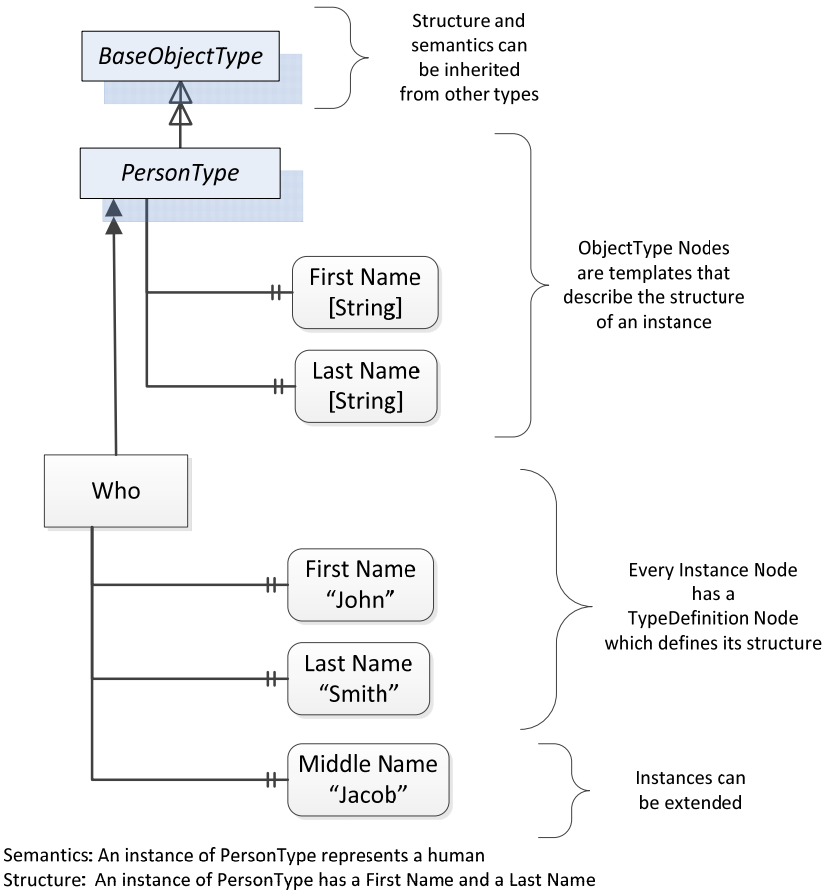

Figure 5 – The Relationship between Type Definitions and Instances

References allow *Nodes* to be connected in ways that describe their relationships. All *References* have a *ReferenceType* that specifies the semantics of the relationship. *References* can be hierarchical or non-hierarchical. Hierarchical references are used to create the structure of *Objects* and *Variables*. Non-hierarchical are used to create arbitrary associations. Applications can define their own *ReferenceType* by creating subtypes of an existing *ReferenceType*. Subtypes inherit the semantics of the parent but may add additional restrictions. Figure 6 depicts several *References,* connecting different *Objects*.

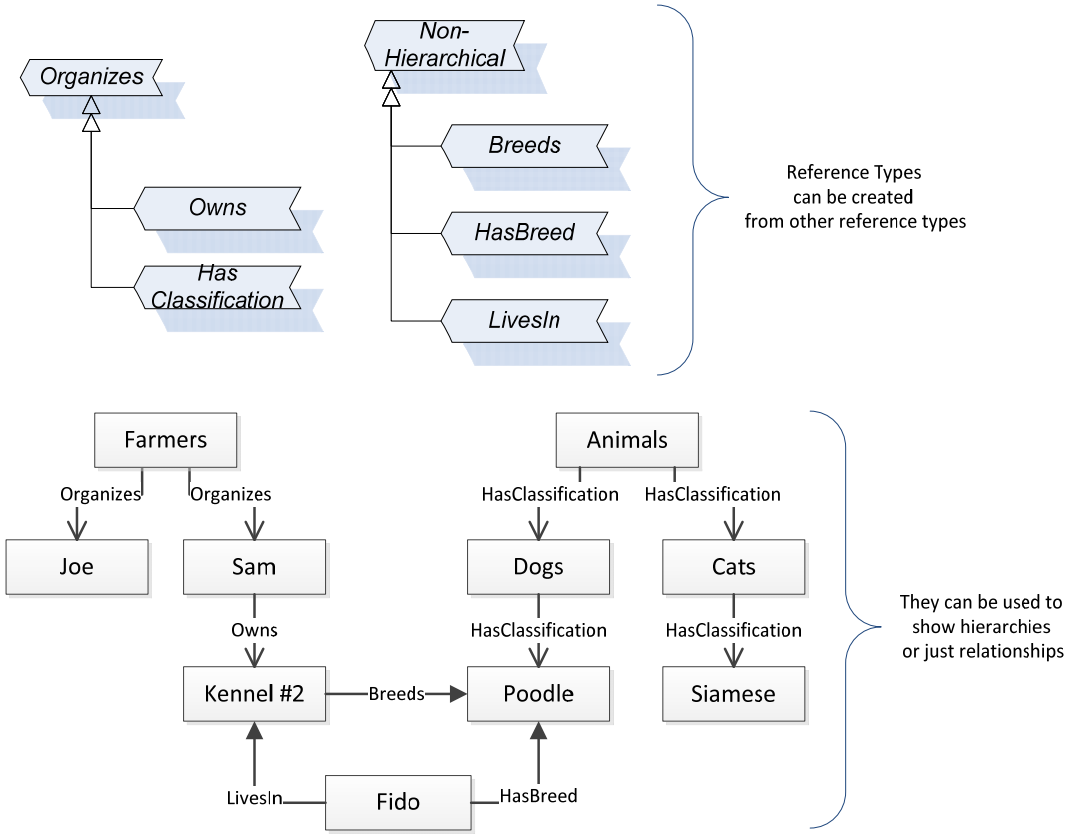

Figure 6 – Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarized in Figure 7. UML representations can also be used; however, the OPC UA notation is less ambiguous because there is a direct mapping from the elements in the figures to *Nodes* in the *AddressSpace* of an OPC UA *Server*.

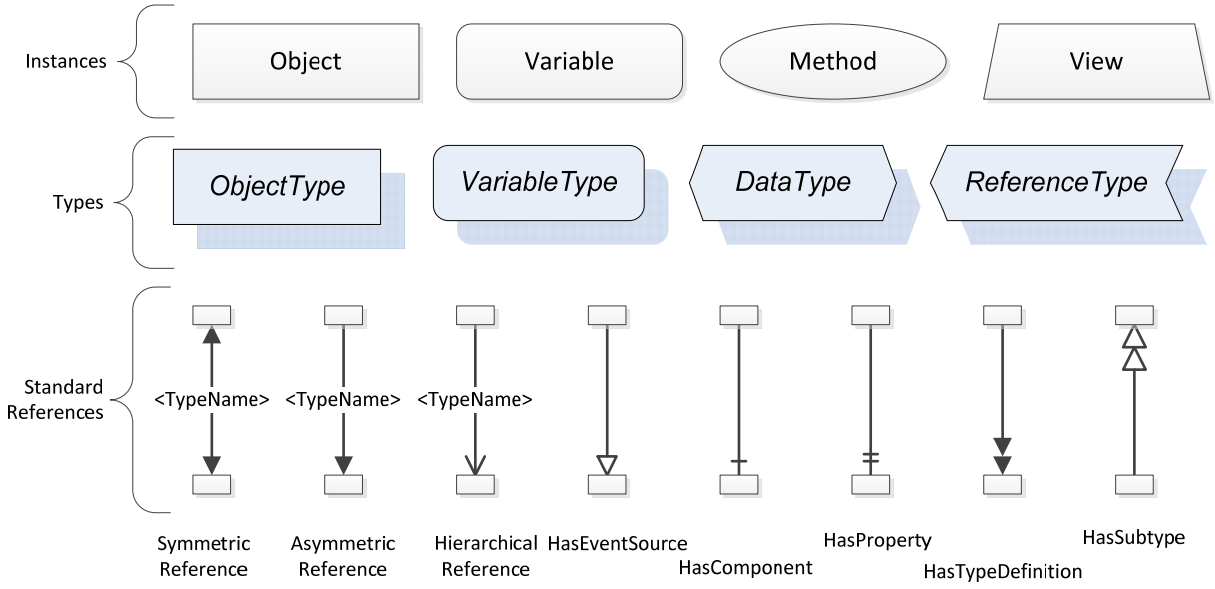

Figure 7 – The OPC UA Information Model Notation

A complete description of the different types of Nodes and References can be found in OPC UA Part 3 and the base structure is described in OPC UA Part 5.

OPC UA specification defines a very wide range of functionality in its basic information model. It is not expected that all *Clients* or *Servers* support all functionality in the OPC UA specifications. OPC UA includes the concept of *Profiles*, which segment the functionality into testable certifiable units. This allows the definition of functional subsets (that are expected to be implemented) within a companion specification. The *Profiles* do not restrict functionality, but generate requirements for a minimum set of functionality (see OPC UA Part 7).

4.2.3.2 Namespaces

OPC UA allows information from many different sources to be combined into a single coherent *AddressSpace*. Namespaces are used to make this possible by eliminating naming and id conflicts between information from different sources. Namespaces in OPC UA have a globally unique string called a NamespaceUri and a locally unique integer called a NamespaceIndex. The NamespaceIndex is only unique within the context of a *Session* between an OPC UA *Client* and an OPC UA *Server*. The *Services* defined for OPC UA use the NamespaceIndex to specify the Namespace for qualified values.

There are two types of values in OPC UA that are qualified with Namespaces: NodeIds and QualifiedNames. NodeIds are globally unique identifiers for *Nodes*. This means the same *Node* with the same NodeId can appear in many *Servers*. This, in turn, means Clients can have built in knowledge of some *Nodes*. OPC UA *Information Models* generally define globally unique *NodeIds* for the *TypeDefinitions* defined by the *Information Model*.

QualifiedNames are non-localized names qualified with a Namespace. They are used for the *BrowseNames* of *Nodes* and allow the same names to be used by different information models without conflict. *TypeDefinitions* are not allowed to have children with duplicate *BrowseNames*; however, instances do not have that restriction.

4.2.3.3 Companion Specifications

An OPC UA companion specification for an industry specific vertical market describes an *Information Model* by defining *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent the concepts used in the vertical market, and potentially also well-defined Objects as entry points into the AddressSpace.

5 Use Cases

Part 1 of this companion specification describes an interface that provides access to asset management and condition monitoring data of motion device systems. Based on the provided data the following use cases are supported:

1) Supervision: With the provided data by the companion specification the robot system can supervisoned and monitored. Functional analysis of individual robot systems within the factory ground is possible. During production phase the companion specification provides data about the operational and safety states as well as process data.

2) Condition monitoring: Condition monitoring is the process of determining the condition of machinery while in operation, in order to identify a significant change which is indicative of a developing fault. This is a major component of Predictive Maintenance where the mainence is scheduled to shorten the downtime. The typical parameters needed for condition monitoring like motor temperature, load, on time are provided by the companion specification for robotics.

3) Asset management: The companion specification for robotics provides detailed information of the main electrical and mechanical parts like part number, brand name, serial number etc. With these data an effective maintanence is possible because the technican knows in adcance which parts are needed exchanged and can be prepared

Figure 8 shows the communication structure with OPC UA.

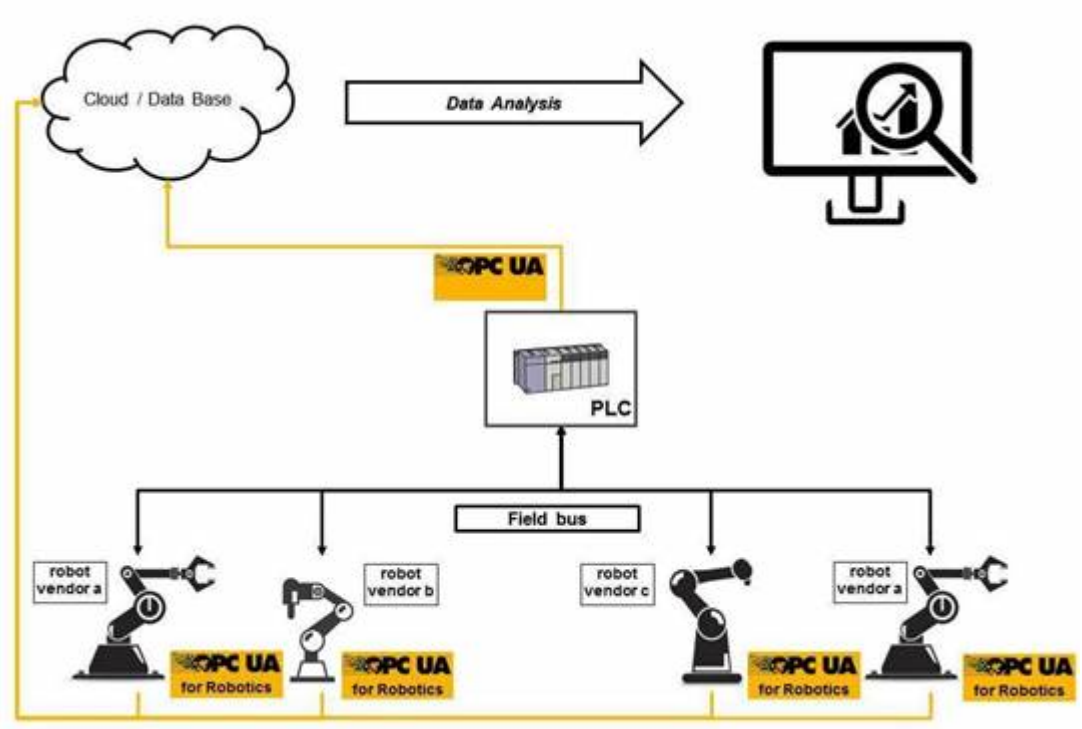

Figure 8 – Communication structure with OPC UA

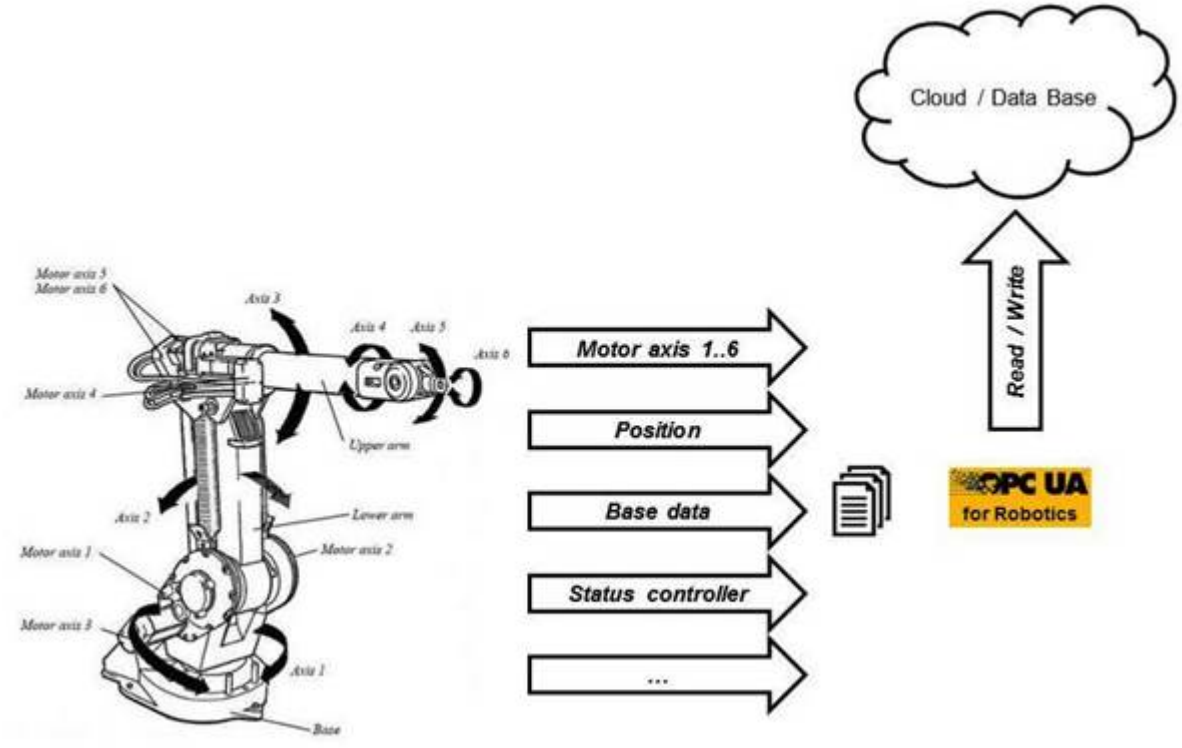

Figure 9 – OPC UA Robotics describes the semantic self-description

6 OPC UA Robotics Information Model overview

The OPC UA (DI) *TopologyElementType* is used as the root object representing the motion device system with all its subcomponents (see Figure 10).

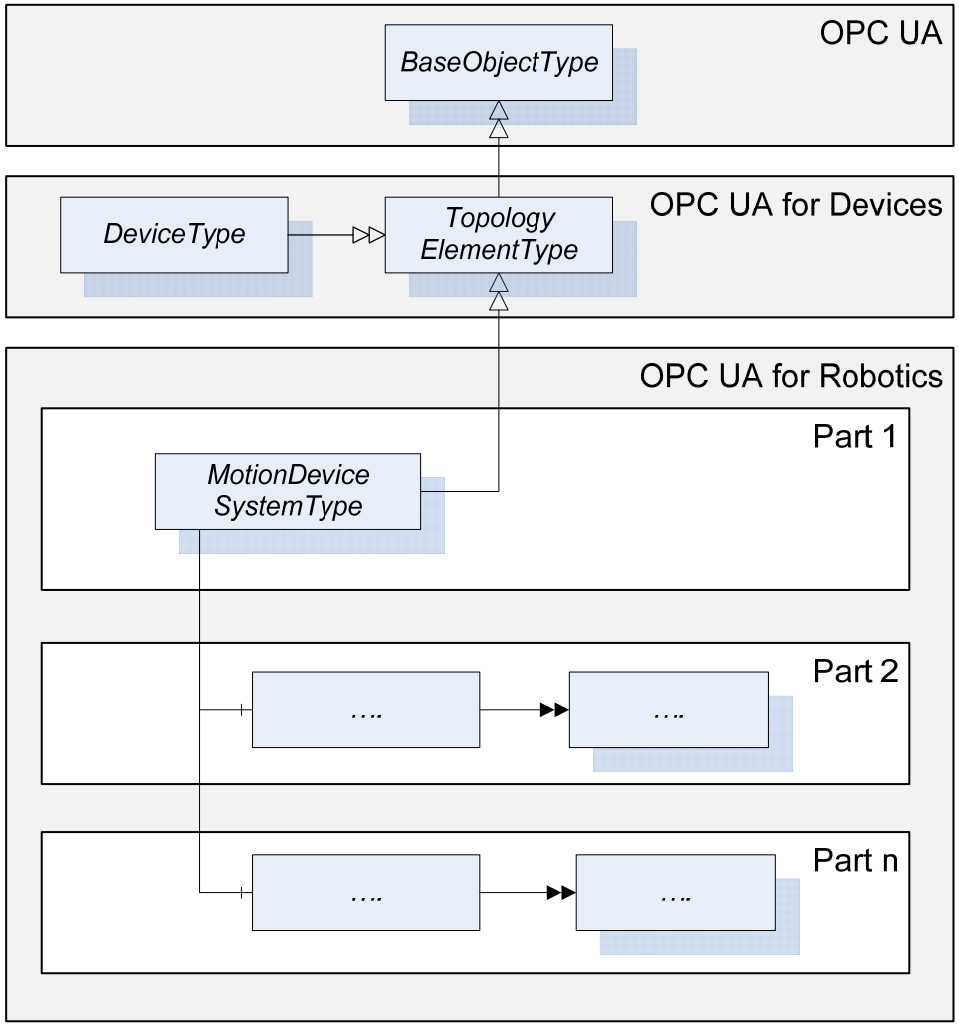

Figure 10 – OPC UA Robotics top level view

Figure 11 shows the main objects and the relations between them. The hatched object types are not finalized in this version.

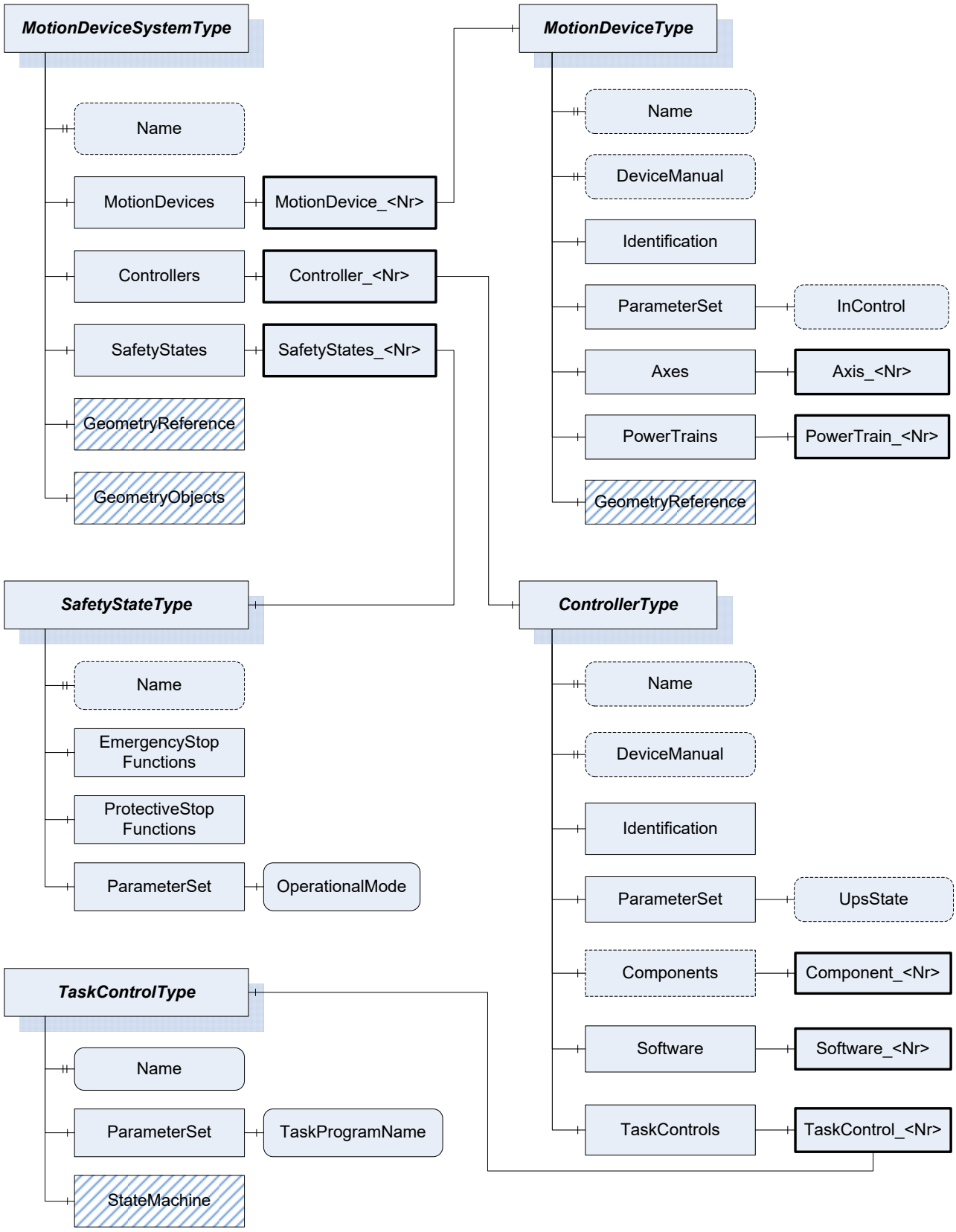

Figure 11 – OPC UA Robotics overview

7 OPC UA ObjectTypes

7.1 MotionDeviceSystemType ObjectType Definition

7.1.1 Overview

The *MotionDeviceSystemType* provides a representation of a motion device system as an entry point to the OPC UA device set. At least one instance of a MotionDeviceSystemType must be instantiated in the *DeviceSet*. This instance organises the information model of a complete robotics system using instances of the described ObjectTypes.

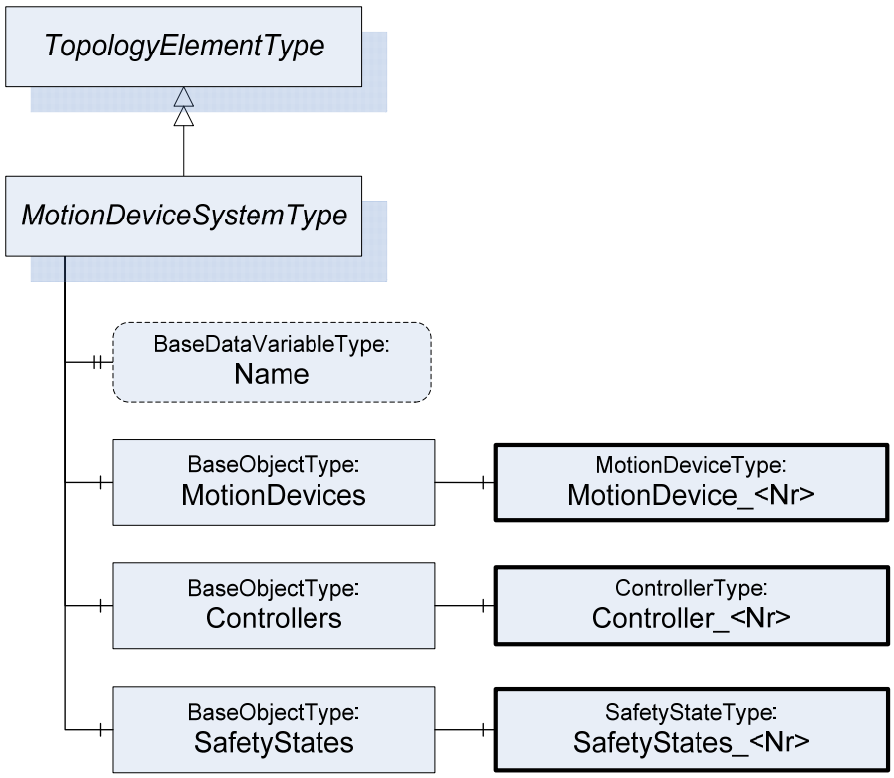

Figure 12 – Overview MotionDeviceSystemType

7.1.2 ObjectType definition

Table 11 – MotionDeviceSystemType Definition

Attribute	**Value**				
BrowseName	MotionDeviceSystemType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
Has Property	Variable	Name	String	BaseDataVariableType	Optional
HasComponent	Object	MotionDevices		BaseObjectType	Mandatory
HasComponent	Object	Controllers		BaseObjectType	Mandatory
HasComponent	Object	SafetyStates		BaseObjectType	Mandatory

Table 12 – TypeDefinition of MotionDevices of MotionDeviceSystemType

Attribute	Value				
BrowseName	MotionDevices				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	MotionDevice_<Nr>		MotionDeviceType	MandatoryPlaceholder

Table 13 – TypeDefinition of Controllers of MotionDeviceSystemType

Attribute	Value				
BrowseName	Controllers				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Controller_<Nr>		ControllerType	MandatoryPlaceholder

Table 14 – TypeDefinition of SafetyStates of MotionDeviceSystemType

Attribute	Value				
BrowseName	SafetyStates				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	SafetyStates_<Nr>		SafetyStateType	MandatoryPlaceholder

7.1.3 ObjectType description

A motion device system may consist of multiple motion devices, controllers and safety systems. References are used to describe the relations between those subsystems. Examples are described in Annex B.1.

7.1.3.1 Variable Name

The *Name* property provides a customer given name for the motion device system or a default name given by the vendor.

7.1.3.2 Object MotionDevices

MotionDevices is a container for one or more instances of the *MotionDeviceType*.

7.1.3.3 Object Controllers

Controllers is a container for one or more instances of the *ControllerType*.

7.1.3.4 Object SafetyStates

SafetyStates is a container for one or more instances of the *SafetyStatesType*.

7.2 MotionDeviceType ObjectType Definition

7.2.1 Overview

The *MotionDeviceType* describes one independent motion device, e. g. a manipulator, a turn table or a linear axis. Examples are described in Annex B.1.

A MotionDevice shall have at least one axis. The *MotionDeviceType is* formally defined in Figure 14.

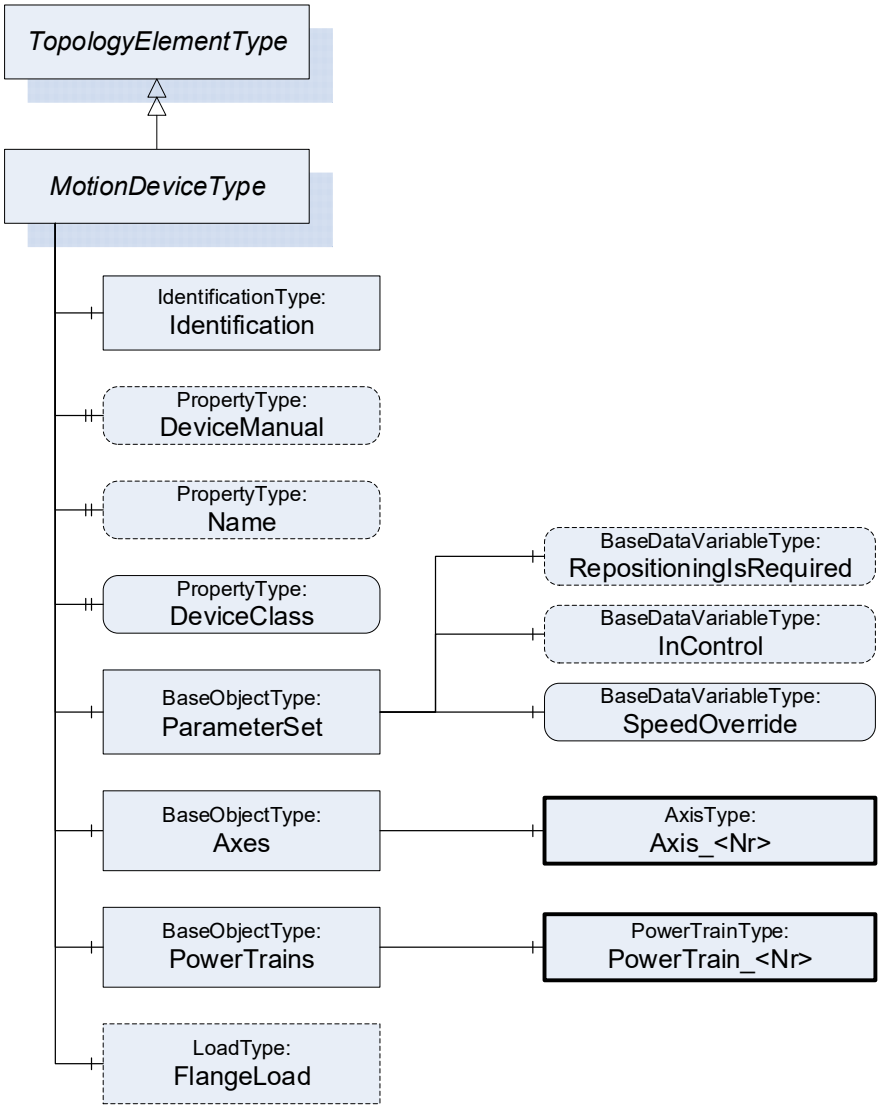

Figure 13 – Overview MotionDeviceType

7.2.2 ObjectType definition

Table 15 – MotionDeviceType Definition

Attribute	**Value**				
BrowseName	MotionDeviceType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	Identification		IdentificationType	Mandatory
HasProperty	Variable	DeviceManual	String	PropertyType	Optional
HasProperty	Variable	Name	String	PropertyType	Optional
HasProperty	Variable	DeviceClass	MotionDeviceClassEnumeration	PropertyType	Mandatory
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory
HasComponent	Object	Axes		BaseObjectType	Mandatory
HasComponent	Object	PowerTrains		BaseObjectType	Mandatory
HasComponent	Object	FlangeLoad		LoadType	Optional

7.2.3 ObjectType description

7.2.3.1 Object Identification

The object *Identification* provides type plate information for identification, see chapter 7.12 for details.

7.2.3.2 Variable DeviceManual

The *DeviceManual* property allows specifying an address of the user manual for the device. It may be a pathname in the file system or a URL (Web address).

7.2.3.3 Variable Name

The *Name* property provides a customer given name for the motion device or a default name given by the vendor.

7.2.3.4 Variable DeviceClass

The variable *DeviceClass* provides the kind of motion device defined by *MotionDeviceClassEnumeration* based on ISO 8373.

Table 16 – MotionDeviceClassEnumeration

MotionDeviceClassEnumeration		
EnumString	Value	Description
OTHER	0	Any motion-device which is not defined by the MotionDeviceClassEnumeration
ARTICULATED_ROBOT	1	This robot design features rotary joints and can range from simple two joint structures to 10 or more joints. The arm is connected to the base with a twisting joint. The links in the arm are connected by rotary joints.
SCARA_ROBOT	2	Robot has two parallel rotary joints to provide compliance in a selected plane
CARTESIAN_ROBOT	3	Cartesian robots have three linear joints that use the Cartesian coordinate system (X, Y, and Z). They also may have an attached wrist to allow for rotational movement. The three prismatic joints deliver a linear motion along the axis.
SPHERICAL_ROBOT	4	The arm is connected to the base with a twisting joint and a combination of two rotary joints and one linear joint. The axes form a polar coordinate system and create a spherical-shaped work envelope.
PARALLEL_ROBOT	5	These spider-like robots are built from jointed parallelograms connected to a common base. The parallelograms move a single end of arm tooling in a dome-shaped work area.
CYLINDRICAL_ROBOT	6	The robot has at least one rotary joint at the base and at least one prismatic joint to connect the links. The rotary joint uses a rotational motion along the joint axis, while the prismatic joint moves in a linear motion. Cylindrical robots operate within a cylindrical-shaped work envelope.

7.2.3.5 Object ParameterSet

Table 17 – ParameterSet of MotionDeviceType

Attribute	**Value**				
BrowseName	AxisType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	OnPath	Boolean	BaseDataVariableType	Optional
HasComponent	Variable	InControl	Boolean	BaseDataVariableType	Optional
HasComponent	Variable	SpeedOverride	Double	BaseDataVariableType	Mandatory

Description of ParameterSet of MotionDeviceType:

– Variable OnPath: The variable *OnPath* is true if the motion device is on or near enough the planned program path such that program execution can continue. If the MotionDevice deviates too much from this path in case of errors or an emergency stop, this value becomes false. If *OnPath* is false, the motion device needs repositioning to continue program execution.

– Variable InControl: The variable *InControl* provides the information if the actuators (in most cases a motor) of the motion device are powered up and in control: "true". The motion device might be in a standstill.

– Variable SpeedOverride: The *SpeedOverride* provides the current speed setting in percent of programmed speed (0 - 100%).

7.2.3.6 Object Axes

Axes is a container for one or more instances of the *AxisType*.

Table 18 – TypeDefinition of Axes of MotionDeviceType

Attribute	**Value**				
BrowseName	Axes				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Axis_<Nr>		AxisType	MandatoryPlaceholder

7.2.3.7 Object PowerTrains

PowerTrains is a container for one or more instances of the *PowerTrainType*.

Table 19 –TypeDefinition of PowerTrains of MotionDeviceType

Attribute	**Value**				
BrowseName	PowerTrains				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	PowerTrain_<Nr>		PowerTrainType	MandatoryPlaceholder

7.2.3.8 Object FlangeLoad

FlangeLoad provides data for the load at the flange or mounting-point of the motion device.

7.2.3.9 Reference IsControlledBy

IsControlledBy is a reference to provide the relationship of motion devices to controllers, if more instances of motion devices or controllers are provided. The *InverseName* is *Controls*.

7.3 AxisType ObjectType Definition

7.3.1 Overview

The *AxisType* describes an axis of a motion device. It is formally defined in Table 20.

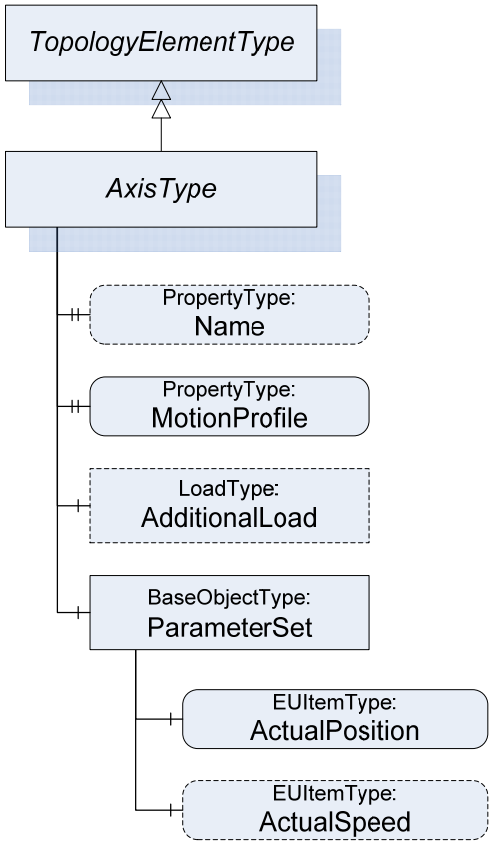

Figure 14 – Overview AxisType

7.3.2 ObjectType definition

Table 20 – AxisType Definition

Attribute	**Value**				
BrowseName	AxisType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	Name	String	PropertyType	Optional
HasProperty	Variable	MotionProfile	AxisMotionProfileEnumeration	PropertyType	Mandatory
HasComponent	Variable	AdditionalLoad		LoadType	Optional
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory

7.3.3 ObjectType description

7.3.3.1 Variable Name

The *Name* property provides a vendor-specifc axis identifier within the control system.

7.3.3.2 Variable MotionProfile

The *MotionProfile* property provides the kind of axis motion as defined by the *AxisMotionProfileEnumeration*.

Table 21 – AxisMotionProfileEnumeration

AxisMotionProfileEnumeration		
EnumString	Value	Description
OTHER	0	Any motion-profile which is not defined by the AxisMotionProfileEnumeration
ROTARY	1	Rotary motion is a rotation along a circular path with defined limits. Motion movement is not going always in the same direction. Control unit is mainly degree.
ROTARY_ENDLESS	2	Rotary motion is a rotation along a circular path with no limits. Motion movement is going endless in the same direction. Control unit is mainly degree.
LINEAR	3	Linear motion is a one dimensional motion along a straight line with defined limits. Motion movement is not going always in the same direction. Control unit is mainly mm.
LINEAR_ENDLESS	4	Linear motion is a one dimensional motion along a straight line with no limits. Motion movement is going endless in the same direction. Control unit is mainly mm.

7.3.3.3 Variable AdditionalLoad

AdditionalLoad provides data for the load that is mounted on this axis, e.g. a transformer for welding.

7.3.3.4 Variable ParameterSet

Table 22 – ParameterSet of AxisType

Attribute	**Value**				
BrowseName	AxisType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	ActualPosition	Double	EUItemType	Mandatory
HasComponent	Variable	ActualSpeed	Double	EUItemType	Optional

Description of ParameterSet of AxisType:

– Variable ActualPosition: The *ActualPosition* variable provides the current position of the axis and may have limits. If the axis has physical limits, the *EURange* property of the *EUItemType* shall be provided.

– Variable ActualSpeed: The *ActualSpeed* variable provides the axis speed. Applicable speed limits of the axis shall be provided by the *EURange* property of the *EUItemType*.

7.3.3.5 Reference IsMovedBy

The *IsMovedBy* reference provides the relationship of axes to power trains. For complex kinematics this does not need to be a one to one relationship, because more than one power train might influence the motion of one axis. Virtual axes that are not actively driven by a power train do not have this reference. The *InverseName* is *Moves*.

7.4 PowerTrainType ObjectType Definition

7.4.1 Overview

The *PowerTrainType* represents instances of power trains of a motion device and is formally defined in Table 23. A power train typically consists of drives, motors and gears to provide the required torque to an axis. Often there is a one-to-one relation between axes and power trains, but it is also possible to have axis coupling and thus one power train can move multiple axes and one axis can be moved by multiple power trains. One power train can have multiple drives, motors and gears when these components move logically the same axes, for example in a master/slave setup. Examples are described in Annex B.1.

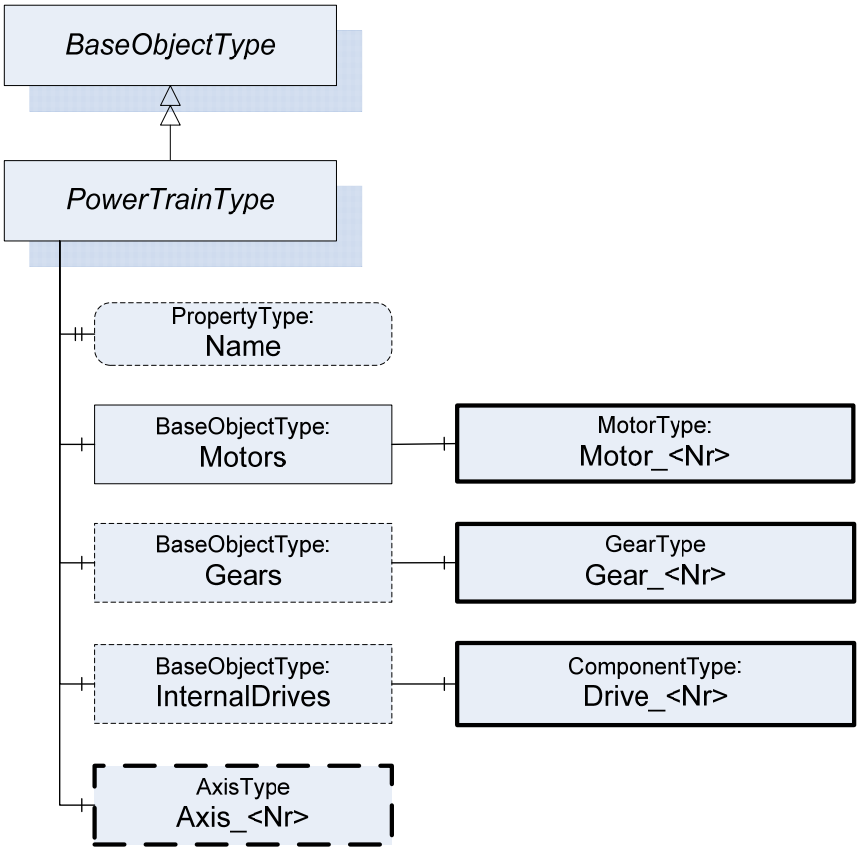

Figure 15 – Overview PowerTrainType

7.4.2 ObjectType definition

Table 23 – PowerTrainType Definition

Attribute	**Value**				
BrowseName	PowerTrainType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasProperty	Variable	Name	String	PropertyType	Optional
HasComponent	Object	Motors		BaseObjectType	Mandatory
HasComponent	Object	Gears		BaseObjectType	Optional
HasComponent	Object	InternalDrives		BaseObjectType	Optional
Moves	Object	Axis_<Nr>		AxisType	OptionalPlaceholder

7.4.3 ObjectType description

7.4.3.1 Variable Name

The *Name* property provides a vendor-specifc power train identifier within the control system.

7.4.3.2 Object Motors

Motors is a container for one or more instances of the *MotorType*.

Table 24 – TypeDefinition of Motors of PowerTrainType

Attribute	**Value**				
BrowseName	Motors				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Motor_<Nr>		MotorType	MandatoryPlaceholder

7.4.3.3 Object Gears

Gears is a container for one or more instances of the *GearType*.

Table 25 – TypeDefinition of Gears of PowerTrainType

Attribute	**Value**				
BrowseName	Gears				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Gear_<Nr>		GearType	MandatoryPlaceholder

7.4.3.4 Object InternalDrives

InternalDrives is a container for one or more instances of the *ComponentType*. Internal drives are located inside the motion device. In contrast, drives located in the cabinet shall be added to the components list of the controller.

Table 26 – TypeDefinition of InternalDrives of PowerTrainType

Attribute	**Value**				
BrowseName	InternalDrives				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	Drive_<Nr>		ComponentType	MandatoryPlaceholder

7.4.3.5 Reference Moves

Moves is a reference to provide the relationship of power trains to axes. The *InverseName* is *IsMovedBy*.

7.5 MotorType ObjectType Definition

7.5.1 Overview

The *MotorType* describes a motor in a power train. It is formally defined in Table 27.

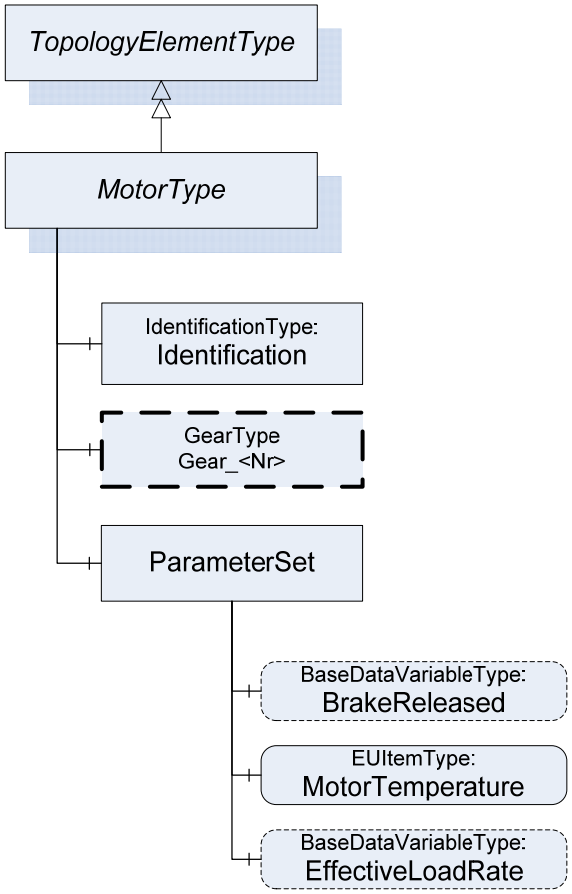

Figure 16 – Overview MotorType

7.5.2 ObjectType definition

Table 27 – MotorType Definition

Attribute	**Value**				
BrowseName	MotorType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	Identification		IdentificationType	Mandatory
IsConnectedTo	Object	Gear_<Nr>		GearType	Optional
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory

7.5.3 ObjectType description

7.5.3.1 Object Idenfication

The object *Identification* provides type plate information for identification, see chapter 7.12 for details.

7.5.3.2 Reference IsDrivenBy

IsDrivenBy is a reference to provide the relationship between a motor and a drive of a power train or a controller.

7.5.3.3 Reference IsConnectedTo

IsConnectedTo is a reference to provide the relationship between a motor and a gear of a power train or a controller.

7.5.3.4 Object ParameterSet

Table 28 – ParameterSet of MotorType

Attribute	**Value**				
BrowseName	MotorType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	BrakeReleased	Boolean	BaseDataVariableType	Optional
HasComponent	Variable	MotorTemperature	Double	EUItemType	Mandatory
HasComponent	Variable	EffectiveLoadRate	UInt16	BaseDataVariableType	Optional

Description of ParameterSet of MotorType:

– Variable *BrakeReleased*: The *BrakeReleased* is an optional variable used only for motors with brakes. If *BrakeReleased* is TRUE the motor is free to run. FALSE means that the motor shaft is locked by the brake.

– Variable *MotorTemperature*: The *MotorTemperature* provides the temperature of the motor. If there is no temperature sensor the value is set to “null”.

– Variable *EffectiveLoadRate*: *EffectiveLoadRate* is expressed as a percentage of maximum continuous load. The Joule integral is typically used to calculate the current load, i.e.:

$$I^2t = \int_{t_0}^{t_1} i^2\,dt$$

Duration should be defined and documented by the vendor.

7.6 GearType ObjectType Definition

7.6.1 Overview

The *GearType* describes a gear in a power train, e.g. a gear box or a spindle. It is formally defined in Table 29.

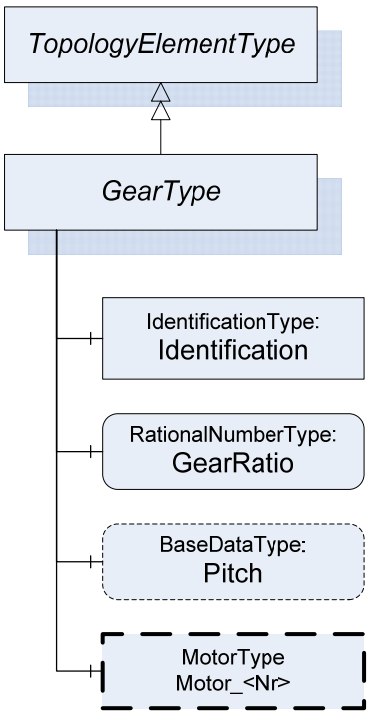

Figure 17 – Overview GearType

7.6.2 ObjectType definition

Table 29 – GearType Definition

Attribute	**Value**				
BrowseName	GearType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	Identification		IdentificationType	Mandatory
HasComponent	Variable	GearRatio	RationalNumber	BaseVariableType	Mandatory
HasComponent	Variable	Pitch	Double	BaseDataType	Optional
IsConnectedTo	Object	Motor_<Nr>		MotorType	Optional

7.6.3 ObjectType description

In case of a one to one relation between powertrains and axes, gear ratio and pitch may reflect the relation between motor and axis velocities. This is not possible when axis coupling is involved because different ratios for all motor-axis combinations may be needed. Additionally, there could be a nonlinear coupling between the

load side of the gear box and the axis. Thus GearRatio and Pitch only reflect the properties of the physical gear box and it may not be possible to use these values to transform between axis and motor movements.

7.6.3.1 Object Identification

The object *Identification* provides type plate information for identification, see chapter 7.12 for details.

7.6.3.2 Object GearRatio

GearRatio is the transmission ratio of the gear expressed as a fraction as input velocity (motor side) by output velocity (load side).

7.6.3.3 Variable Pitch

Pitch describes the distance covered in millimeters (mm) for linear motion per one revolution of the output side of the driving unit. *Pitch* is used in combination with *GearRatio* to describe the overall transmission from input to output of the gear.

Calculation formula:

$$Linear\ distance = \frac{Revolutions\ of\ input}{GearRatio} \times Pitch$$

7.6.3.4 Reference IsConnectedTo

IsConnectedTo is a reference to provide the relationship between a motor and a gear of a power train or a controller.

7.7 SafetyStateType ObjectType Definition

7.7.1 Overview

SafetyStateType describes the safety states of the motion devices and controllers. One motion device system is associated with one or more instances of the *SafetyStateType*.

The *SafetyStateType* was modelled directly in the *MotionDeviceSystemType* for the following reasons:

- The manufacturers of systems have different concepts where safety is functional located, e.g. the hardware and software implementation.
- The safety state typically applies to the entire robotic system. If multiple safety state instances are implemented in robotic systems, these can be represented by individual instances of the *SafetyStateType* and associated with the controller by reference.

The safety state is for informational purpose only and not intended for use with functional safety applications as defined in ISO 61508.

The *SafetyStateType* is formally defined in Table 30.

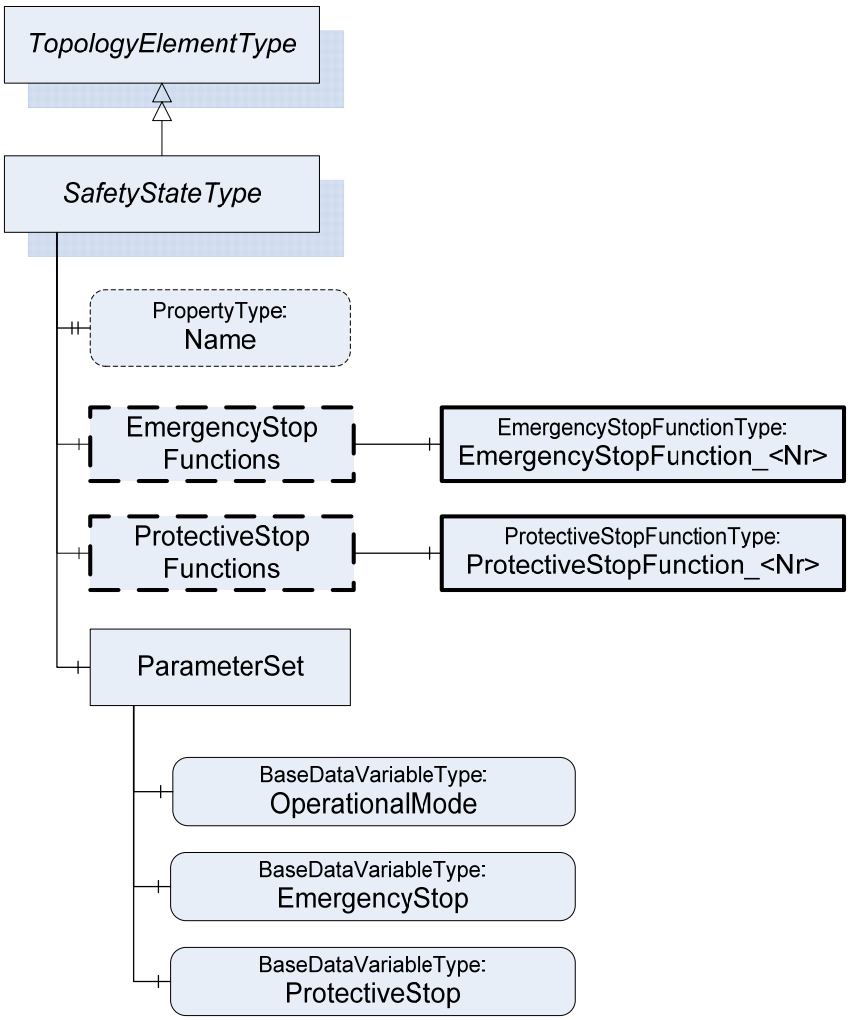

Figure 18 – Overview SafetyStateType

7.7.2 ObjectType definition

Table 30 – SafetyStateType Definition

Attribute	**Value**				
BrowseName	SafetyStateType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	Name	String	PropertyType	Optional
HasComponent	Object	EmergencyStopFunctions		BaseObjectType	Optional
HasComponent	Object	ProtectiveStopFunctions		BaseObjectType	Optional
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory

7.7.3 ObjectType description

7.7.3.1 Variable Name

The *Name* property provides a vendor-specifc safety state identifier within the motion device system.

7.7.3.2 Object EmergencyStopFunctions

EmergencyStopFunctions is a container for one or more instances of the *EmergencyStopFunctionType*. The number and names of emergency stop functions is vendor specific. When provided, this object contains a list of all emergency stop functions with names and current state. See description of *EmergencyStopFunctionType* for examples of emergency stop functions.

Table 31 – TypeDefinition of EmergencyStopFunctions of SafetyStateType

Attribute	**Value**				
BrowseName	EmergencyStopFunctions				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	EmergencyStopFunction_<Nr>		EmergencyStopFunctionType	Mandatory Placeholder

Table 32 – ObjectType EmergencyStopFunctionType

Attribute	**Value**				
BrowseName	EmergencyStopFunctionType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasProperty	Variable	Name	String	PropertyType	Mandatory
HasComponent	Variable	Active	Boolean	BaseDataVariableType	Mandatory

Description of *EmergencyStopFunctionType*:

According to ISO 10218-1:2011 Ch.5.5.2 Emergency stop the robot shall have one or more emergency stop functions.

– Variable *Name*: The *Name* property provides a vendor-specifc emergency stop function identifier within the safety system. The only named emergency stop function in the ISO 10218-1:2011 standard is the "Pendant emergency stop function". Other than that, the standard does not give any indication on naming of emergency stop functions.

– Variable *Active*: *Active* is TRUE if this particular emergency stop function is active, e.g. that the emergency stop button is pressed, FALSE otherwise.

7.7.3.3 Object ProtectiveStopFunctions

ProtectiveStopFunctions is a container for one or more instances of the *ProtectiveStopFunctionType*. The number and names of protective stop functions is vendor specific. When provided, this object contains a list of all protective stop functions with names and current state. See description of *ProtectiveStopFunctionType* for examples of protective stop functions.

Table 33 – TypeDefinition of ProtectiveStopFunctions of SafetyStateType

Attribute	**Value**				
BrowseName	ProtectiveStopFunctions				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	ProtectiveStopFunction_<Nr>		ProtectiveStopFunctionType	Mandatory Placeholder

Table 34 – ObjectType ProtectiveStopFunctionType

Attribute	**Value**				
BrowseName	ProtectiveStopFunctionType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasProperty	Variable	Name	String	PropertyType	Mandatory
HasComponent	Variable	Enabled	Boolean	BaseDataVariableType	Mandatory
HasComponent	Variable	Active	Boolean	BaseDataVariableType	Mandatory

Description of ProtectiveStopFunctionType:

According to ISO 10218-1:2011 Ch.5.5.3 the robot shall have one or more protective stop functions designed for the connection of external protective devices.

– Variable *Name*: The *Name* property provides a vendor-specifc protective stop function identifier within the safety system.

– Variable *Enabled*: Enabled is TRUE if this protective stop function is currently supervising the system, FALSE otherwise. A protective stop function may or may not be enabled at all times, e.g. the protective stop function of the safety doors are typically enabled in automatic operational mode and disabled in manual mode. On the other hand for example, the protective stop function of the teach pendant enabling device is enabled in manual modes and disabled in automatic modes.

– Variable *Active*: *Active* is TRUE if this particular protective stop function is active, i.e. that a stop is initiated, FALSE otherwise. If *Enabled* is FALSE then *Active* shall be FALSE.

Examples

The table below shows an example with a door interlock function. In this example, the door is only monitored during automatic modes. During manual modes, the operators may open the door without causing a protective stop.

Table 35 – Door Interlock Protective Stop Example

	Automatic Mode		**Manual Mode**	
Door interlock	**Enabled**	**Active**	**Enabled**	**Active**
Door closed	TRUE	FALSE	FALSE	FALSE
Door open	TRUE	TRUE	FALSE	FALSE

The next example shows how the three-position enabling device normally found on teach pendants is processed. In this case it does not matter if the enabling device is pressed or not during automatic modes, while in manual modes, a protective stop is active as long as the enabling device is released or fully pressed.

Table 36 – Teach Pendant Enabling Device Protective Stop Example

	Automatic Mode		**Manual Mode**	
Teach Pendant Enabling Device	**Enabled**	**Active**	**Enabled**	**Active**
Released	FALSE	FALSE	TRUE	TRUE
Middle position	FALSE	FALSE	TRUE	FALSE
Fully pressed (panic)	FALSE	FALSE	TRUE	TRUE

7.7.3.4 ParameterSet

Table 37 – ParameterSet of SafetyStateType

Attribute	**Value**				
BrowseName	ParameterSet				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	OperationalMode	OperationalModeEnumeration	BaseDataVariableType	Mandatory
HasComponent	Variable	EmergencyStop	Boolean	BaseDataVariableType	Mandatory
HasComponent	Variable	ProtectiveStop	Boolean	BaseDataVariableType	Mandatory

Description of *ParameterSet* of *SafetyStateType*:

– Variable *OperationalMode*: *OperationalMode* provides information about the current operational mode. Allowed values are described in *OperationalModeEnumeration*, see ISO 10218-1:2011 Ch.5.7 Operational Modes.

– Variable *EmergencyStop*: *EmergencyStop* is TRUE if one or more of the emergency stop functions in the robot system are active, FALSE otherwise. If the *EmergencyStopFunctions* object is provided, then the value of this variable is TRUE if one or more of the listed emergency stop functions are active.

– Variable *ProtectiveStop*: *ProtectiveStop* is TRUE if one or more of the enabled protective stop functions in the system are active, FALSE otherwise. If the *ProtectiveStopFunctions* object is provided, then the value of this variable is TRUE if one or more of the listed protective stop functions are enabled and active.

Table 38 – OperationalModeEnumeration

OperationalModeEnumeration		
EnumString	*Value*	*Description*
OTHER	0	This value is used when there is no valid operational mode. Examples are: - During system-boot - The system is not calibrated (and hence can not verify cartesian position values) - There is a failure in the safety system itself
MANUAL_REDUCED_SPEED	1	"Manual reduced speed" - name according to ISO 10218-1:2011
MANUAL_HIGH_SPEED	2	"Manaul high speed" - name according to ISO 10218-1:2011
AUTOMATIC	3	"Automatic" - name according to ISO 10218-1:2011
AUTOMATIC_EXTERNAL	4	"Automatic external" - Same as "Automatic" but with external control, e.g. by a PLC

7.8 ControllerType ObjectType Definition

7.8.1 Overview

The *ControllerType* describes the control unit of motion devices. One motion device system can have one or more instances of the *ControllerType*. The *ControllerType* is formally defined in Table 39.

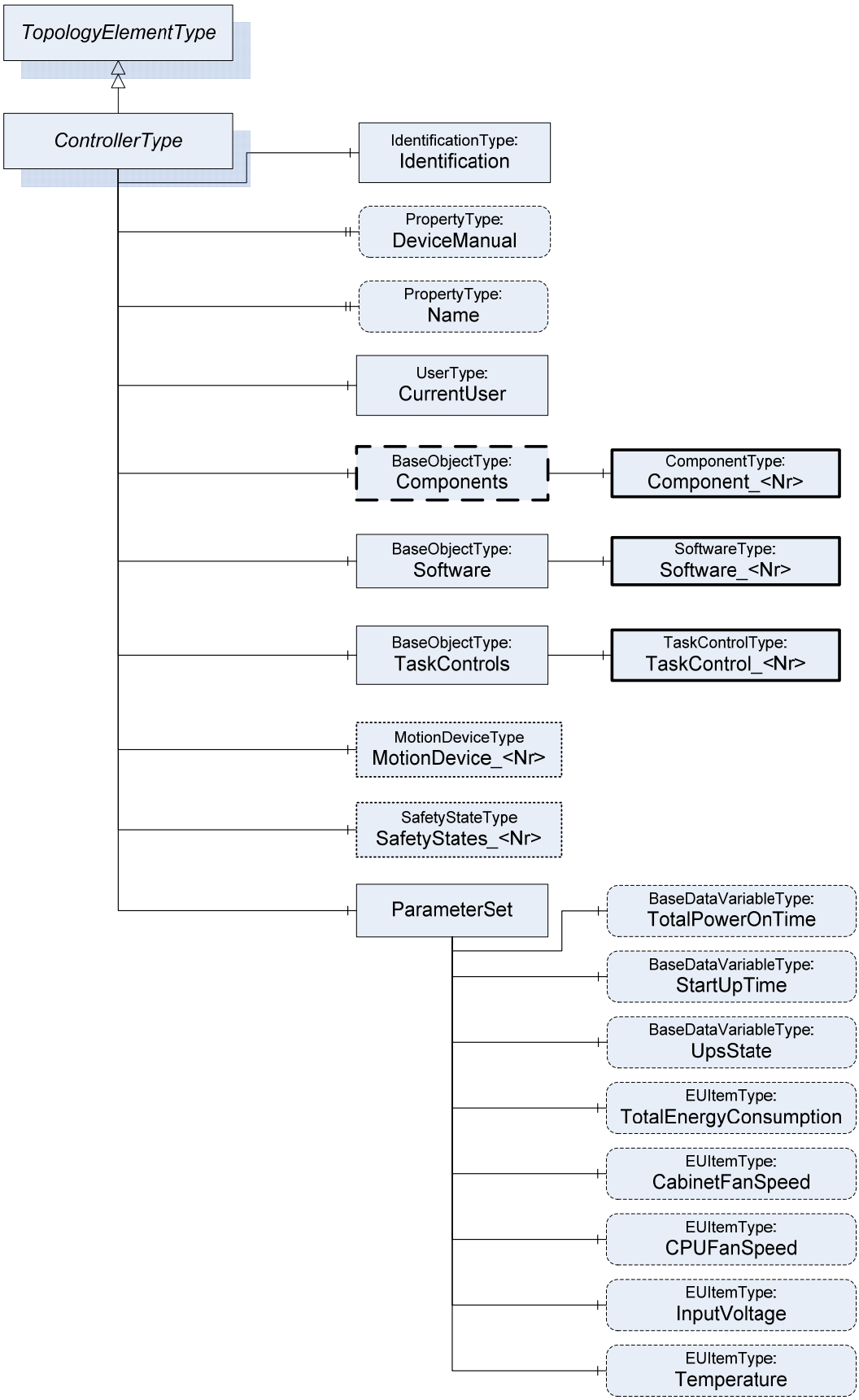

Figure 19 – Overview ControllerType

7.8.2 ObjectType definition

Table 39 – ControllerType Definition

Attribute	Value				
BrowseName	ControllerType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	Identification		IdentificationType	Mandatory
HasProperty	Variable	DeviceManual	String	PropertyType	Optional
HasProperty	Variable	Name	String	PropertyType	Optional
HasComponent	Variable	CurrentUser		UserType	Mandatory
HasComponent	Object	Components		BaseObjectType	Optional
HasComponent	Object	Software		BaseObjectType	Mandatory
HasComponent	Object	TaskControls		BaseObjectType	Mandatory
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory
HasSafetyStates	Object	SafetyStates_<Nr>		SafetyStateType	OptionalPlaceholder
Controls	Object	MotionDevice_<Nr>		MotionDeviceType	OptionalPlaceholder

7.8.3 ObjectType description

7.8.3.1 Object Identification

The object *Identification* provides type plate information for identification, see chapter 7.12 for details.

7.8.3.2 Variable DeviceManual

The *DeviceManual* property allows specifying an address of the user manual for the controller. It may be a pathname in the file system or a URL (Web address).

7.8.3.3 Variable Name

The *Name* property provides a customer given name for the controller or a default name given by the vendor.

7.8.3.4 Variable CurrentUser

The variable *CurrentUser* provides information about the active vendor specific user level of the controller.

7.8.3.5 Object Components

Components is a container for one or more instances of *ComponentType*. The listed components are installed in the controller, e.g. a processing-unit, a power-supply or a drive.

Table 40 – TypeDefinition of Components of ControllerType

Attribute	Value				
BrowseName	Components				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	Component_<Nr>		ComponentType	MandatoryPlaceholder

7.8.3.6 Object Software

Software is a container for one or more instances of *SoftwareType*. Each controller has at least one software installed.

Table 41 – TypeDefinition of Software of ControllerType

Attribute	Value				
BrowseName	Software				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	Software _<Nr>		SoftwareType	MandatoryPlaceholder

7.8.3.7 Object TaskControls

TaskControls is a container for one or more instances of TaskControlType.

Table 42 – TypeDefinition of TaskControls of ControllerTyp

Attribute	Value				
BrowseName	TaskControls				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	TaskControl _<Nr>		TaskControlType	MandatoryPlaceholder

7.8.3.8 ParameterSet

Table 43 – ParameterSet of ControllerType

Attribute	Value				
BrowseName	ControllerType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	TotalPowerOnTime	DurationString	BaseDataVariableType	Optional
HasComponent	Variable	StartUpTime	DateTime	BaseDataVariableType	Optional
HasComponent	Variable	UpsState	String	BaseDataVariableType	Optional
HasComponent	Variable	TotalEnergyConsumption	Double	EUItemType	Optional
HasComponent	Variable	CabinetFanSpeed	Double	EUItemType	Optional
HasComponent	Variable	CPUFanSpeed	Double	EUItemType	Optional
HasComponent	Variable	InputVoltage	Double	EUItemType	Optional
HasComponent	Variable	Temperature	Double	EUItemType	Optional

Description of *ParameterSet* of *ControllerType*:

– Variable *TotalPowerOnTime*: *TotalPowerOnTime* provides the total accumulated time the controller was powered on.

– Variable *StartUpTime*: *StartUpTime* provides the date and time of the last start-up of the controller.

– Variable *UpsState*: *UpsState* provides the vendor specific status of an integrated uninterruptible power supply or accumulator system.

– Variable *TotalEnergyConsumption*: *TotalEnergyConsumption* provides total accumulated energy consumed by the motion devices related with this controller instance.

– Varibale *CabinetFanSpeed*: *CabinetFanSpeed* provides the speed of the cabinet fan.

– Variable *CPUFanSpeed*: *CPUFanSpeed* provides the speed of the CPU fan.

– Variable *InputVoltage*: *InputVoltage* provides the input voltage of the controller which can be a configured value. To distuinguish between a AC or DC supply the optional property *Definition* of the base type *DataItemType* shall be used.

– Variable *Temperature*: *Temperature* provides the controller temperature given by a temperature sensor inside of the controller.

7.8.3.9 Reference HasSafetyStates

The *HasSafetyStates* reference provides the relationship of safety states to a controller.

7.8.3.10 Reference Controls

The *Controls* reference provides the relationship of a motion device and controller. The InverseName is IsControlledBy.

7.9 ComponentType ObjectType Definition

7.9.1 Overview

The *ComponentType* describes components mounted in the controller cabinet. It is formally defined in Table 44.

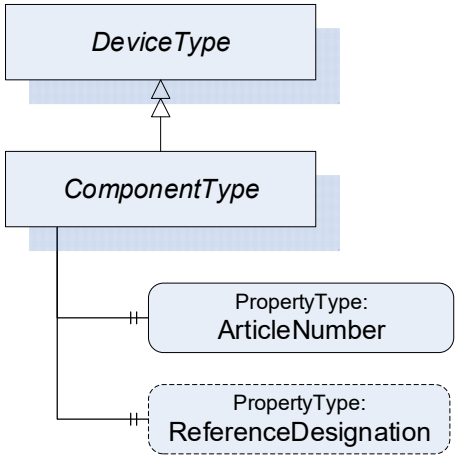

Figure 20 – Overview ComponentType

7.9.2 ObjectType definition

Table 44 – ComponentType Definition

Attribute	**Value**				
BrowseName	ComponentType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the DeviceType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	ArticleNumber	String	PropertyType	Mandatory
HasProperty	Variable	ReferenceDesignation	String	PropertyType	Optional

7.9.3 ObjectType description

7.9.3.1 Variable ArticleNumber

The *ArticleNumber* property provides order information usually displayed on type shields or in ERP systems.

7.9.3.2 Variable ReferenceDesignation

The *ReferenceDesignation* property provides information about the location of the device. It should allow to find the device in the real robot system. Typically it also allows to find the device in an electric scheme (typically EN 81346-2 is used) and get even more informations from the device and the location. A vendor could provide the reference designation strings from the electric schema. If there is no reference designation as it could be for none electrical devices the entry can be skipped or left empty.

An use case could be to build up a location oriented view in a spare part management client software. It enables to identify parts with the same article number which is not possible if this entry is not used.

7.10 SoftwareType ObjectType Definition

7.10.1 Overview

The ObjectType *SoftwareType* defines the structure of a Software Object. Figure 21 shows the *SoftwareType*. It is formally defined Table 45.

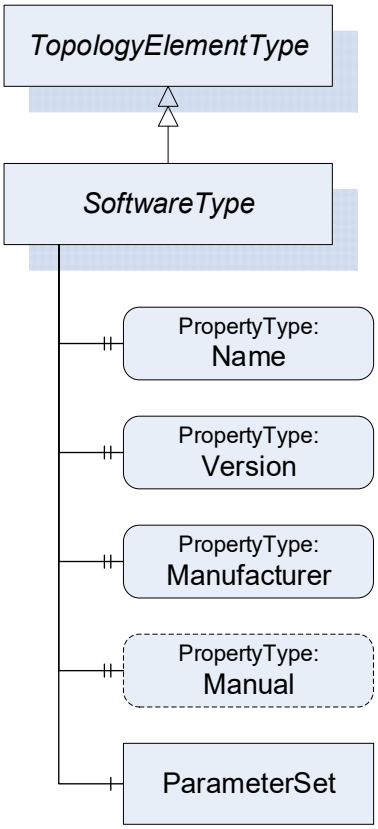

Figure 21 – Overview SoftwareType

7.10.2 ObjectType definition

The *SoftwareType* represents instances of software installed on the controller.

Table 45 – SoftwareType Definition

Attribute	Value				
BrowseName	SoftwareType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	Name	String	PropertyType	Mandatory
HasProperty	Variable	Version	String	PropertyType	Mandatory
HasProperty	Variable	Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Manual	String	PropertyType	Optional
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory

7.10.3 ObjectType description

7.10.3.1 Variable Name

The *Name* Property provides the vendor given name of the software.

7.10.3.2 Variable Version

The *Version* Property provides the vendor version of the software.

7.10.3.3 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the software.

7.10.3.4 Variable Manual

The *Manual* property allows specifying an address of the user manual for the software. It may be a pathname in the file system or a URL (web address).

7.10.3.5 Object ParameterSet

The ParameterSet is for vendor specific extension of information about the software. ParameterSet variables should be extended in a vendor specific namespace. A BuildDate or InstallationDate might be meaningfull information.

7.11 TaskControlType ObjectType Definition

7.11.1 Overview

The *TaskControlType* represents instances of task controls of a controller and is formally defined in Table 46.

The task control describes an execution engine that loads and runs task programs. One task runs one task program at the time. The system should instantiate the maximum allowed number of task controls.

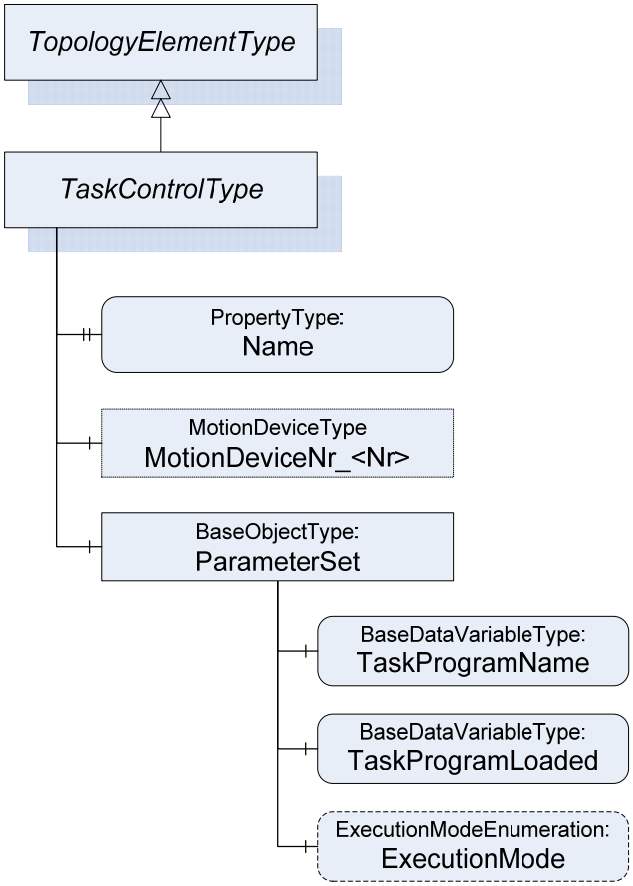

Figure 22 – Overview TaskControlType

7.11.2 ObjectType definition

Table 46 – TaskControlType Definition

Attribute	**Value**				
BrowseName	TaskControlType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TopologyElementType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	Name	String	PropertyType	Mandatory
HasComponent	Object	ParameterSet		BaseObjectType	Mandatory
Controls	Object	MotionDevice_<Nr>		MotionDeviceType	OptionalPlaceholder

7.11.3 ObjectType description

7.11.3.1 Variable Name

The *Name* property provides a customer given identifier for the task control or a default name given by the vendor.

7.11.3.2 Object ParameterSet

Table 47 – ParameterSet of TaskControlType

Attribute	**Value**				
BrowseName	ParameterSet				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	TaskProgramName	String	BaseDataVariableType	Mandatory
HasComponent	Variable	TaskProgramLoaded	Boolean	BaseDataVariableType	Mandatory
HasComponent	Variable	ExecutionMode	Enumeration	ExecutionModeEnumeration	Optional

Description of *ParameterSet* of *TaskControlType*:

– Variable *TaskProgramName*: The *TaskProgramName* property provides a customer given identifier for the task program.

– Variable *TaskProgramLoaded*: TaskProgramLoaded is TRUE if a task program is loaded in the task control, FALSE otherwise.

– Variable *ExecutionMode*: The ExecutionMode tells how the task control executes the task program.

Table 48 – ExecutionModeEnumeration

ExecutionModeEnumeration		
EnumString	*Value*	*Description*
CYCLE	0	Single execution of a task program according to ISO 8373
CONTINUOUS	1	Task program is executed continuously and starts again automatically
STEP	2	Task program is executed in steps

7.11.3.3 Reference Controls

Controls is a reference to provide the relationship between a task control and a motion device.

7.12 IdentificationType ObjectType Definition

7.12.1 Overview

The *IdentificationType* provides identification information, e.g. which are shown on a type shield, of devices and components. It is formally defined in Table 49.

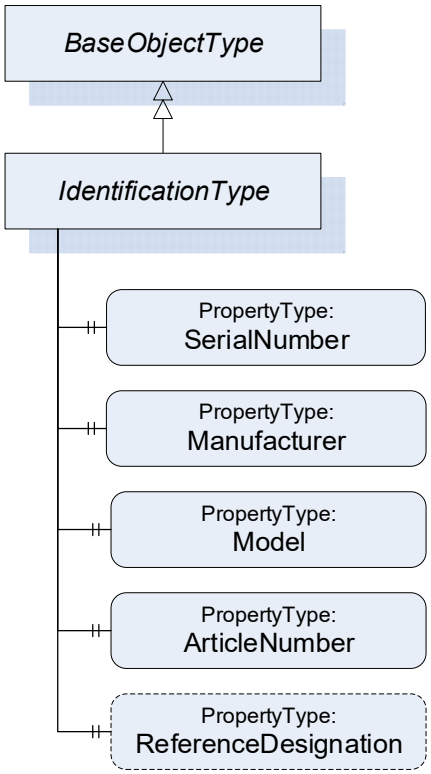

Figure 23 – Overview IdentificationType

7.12.2 ObjectType definition

Table 49 – IdentificationType Definition

Attribute	Value				
BrowseName	IdentificationType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasProperty	Variable	SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	ArticleNumber	String	PropertyType	Mandatory
HasProperty	Variable	ReferenceDesignation	String	PropertyType	Optional

7.12.3 ObjectType description

7.12.3.1 Variable SerialNumber

The *SerialNumber* property is a unique production number assigned by the manufacturer of the device. This is often stamped on the outside of the device and may be used for traceability and warranty purposes.

7.12.3.2 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the device.

7.12.3.3 Variable Model

The *Model* property provides the brand name of the device.

7.12.3.4 Variable ArticleNumber

The *ArticleNumber* property provides order information usually displayed on type shields or in ERP systems.

7.12.3.5 Variable ReferenceDesignation

The *ReferenceDesignation* property provides information about the location of the device. It should allow to find the device in the real robot system. Typically it also allows to find the device in an electric scheme and get even more information about the device and its location. Therefore a vendor could provide the reference designation string used in the electric schema. If there is no reference designation the entry can be skipped or left empty. This could be the case for non-electrical devices.

A use case for the *ReferenceDesignation* could be to build up a location oriented view in a spare part management client software. It enables to clearly identify parts with the same article number which is not possible if this entry is not used.

7.13 LoadType ObjectType Definition

7.13.1 Overview

The *LoadType* is for describing loads mounted on the motion device typically by an integrator or a customer and is formally defined in Table 50. Instances of this *ObjectType* definition are used to describe the load mounted on one of several mounting points. A very common mounting point is the flange of a motion device. Typically a motion device has additional mounting points on some of the axis. The provided values can either be determined by the robot controller or can be set up by an operator.

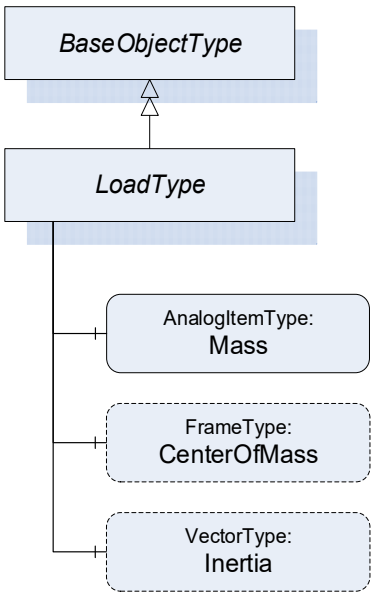

Figure 24 – Overview LoadType

7.13.2 ObjectType definition

Table 50 – LoadType Definition

Attribute	Value				
BrowseName	LoadType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasComponent	Variable	Mass	Double	AnalogItemType	Mandatory
HasComponent	Variable	CenterOfMass	Frame	FrameType	Optional
HasComponent	Variable	Inertia	Vector	VectorType	Optional

7.13.3 ObjectType description

7.13.3.1 Variable Mass

The variable *Mass* provides the weight of the load mounted on one mounting point.

The *EngineeringUnits* of the *Mass* shall be provided.

7.13.3.2 Variable CenterOfMass

The variable *CenterOfMass* provides the position and orientation of the center of the mass related to the mounting point using a FrameType. X, Y, Z define the position of the center of gravity relative to the mounting point coordinate system. A, B, C define the orientation of the principal axes of inertia relative to the mounting point coordinate system. Orientation A, B, C can be "0" for systems which do not need this values.

If the instance of the *LoadType* describes the flange load of a motion device the mounting point coordinate system is the flange coordinate system defined in the *GeometryReference*. If the instance of the *LoadType* describes an additional load of an axis the mounting point coordinate system is vendor specific and it is up to the vendor to model this coordinate system in the *GeometryReference*.

7.13.3.3 Variable Inertia

The variable *Inertia* uses the *VectorType* to describe the three values of the principal moments of inertia with respect to the mounting point coordinate system. If inertia values are provided for rotary axis the *CenterOfMass* shall be completely filled as well. Table 51 describes the possible degrees of modelling from a minimal one e.g. only the weight of the mass to a complete one comprising weight, center of mass, principal axes and inertia.

Table 51 – LoadType possible degrees of modelling

	Mass	CenterOfMass		Inertia
		X, Y, Z	A, B, C	
Mass only	Used	-	-	-
Mass with center of gravity	Used	Used	0, 0, 0	-
Mass with inertia	Used	Used	Used	Used

7.14 UserType ObjectType Definition

7.14.1 Overview

The *UserType ObjectType* describes information of the registred user groups within the control system.

It is formally defined in Table 52.

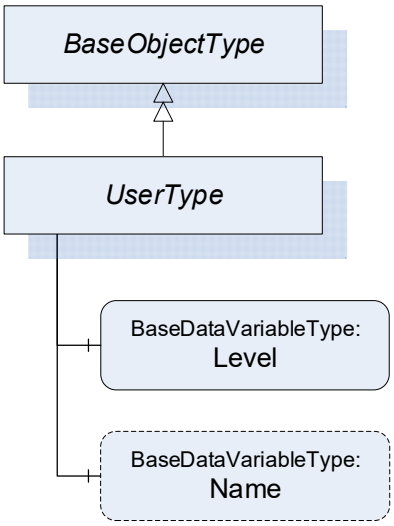

Figure 25 – Overview UserType

7.14.2 ObjectType definition

Table 52 – UserType Definition

Attribute	**Value**				
BrowseName	UserType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in OPC Unified Architecture					
HasProperty	Variable	Level	String	BaseDataVariableType	Mandatory
HasProperty	Variable	Name	String	BaseDataVariableType	Optional

7.14.3 ObjectType description

7.14.3.1 Variable Level

The *Level* property provides information about the access rights and determines what can be viewed, updated or deleted by a user. Depending on the user level different functionalities are available. The robot vendors might use different descriptions and access levels for the users and might require authentification.

7.14.3.2 Variable Name

The *Name* property provides the name for the current user within the control system.

8 OPC UA ReferenceTypes

8.1 General

This section defines the ReferenceTypes that are inherent to the present companion specification. Figure 26 describes informally the hierarchy of these Reference Types. OPC UA Reference Types are defined in Part 3.

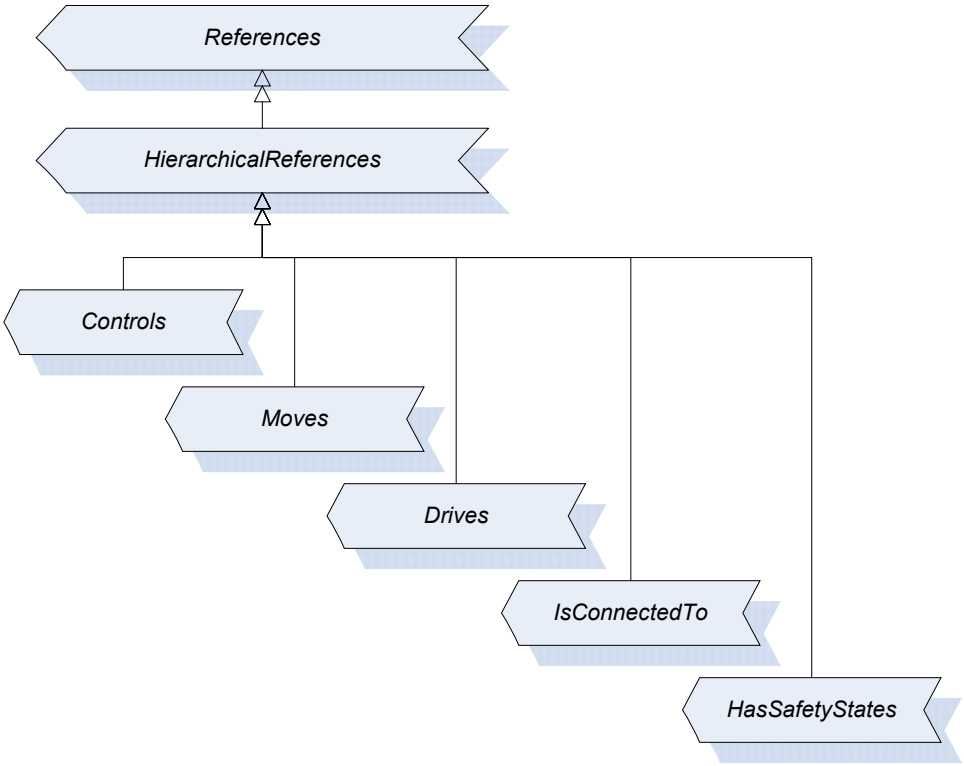

Figure 26 – Reference Type Hierarchy

8.2 Controls (IsControlledBy) Reference Type

The OPC UA *ReferenceType Controls* is used to describe dependencies between objects which have a controlling character. The *BrowseName Controls* and the *InverseName IsControlledBy* describe semantically the hierarchical dependency e.g. a controlling device *Controls* a controlled machine module.

Example for usage in this companion specification: If one controller Controls several motion devices, each motion device IsControlledBy the same controller.

Table 53 – Controls Reference Definition

Attributes	**Value**				
BrowseName	Controls				
InverseName	IsControlledBy				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferenceType defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.3 Moves (IsMovedBy) Reference Type

Table 54 – Controls Reference Definition

Attributes	**Value**				
BrowseName	Moves				
InverseName	IsMovedBy				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferenceType defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.4 Drives (IsDrivenBy) Reference Type

The OPC UA *ReferenceType* Drives is used to describe dependencies between objects which have a driving or powering character. The BrowseName Drives and the InverseName IsDrivenBy describe semantically the hierarchical dependency.

Example for usage in this companion specification: an electrical motor IsDrivenBy an servo amplifier (drive) and an internal drive of a motion device or a drive as a component of a controller Drives a motor.

Table 55 – Drives Reference Definition

Attributes	**Value**				
BrowseName	Drives				
InverseName	IsDrivenBy				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferenceType defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.5 IsConnectedTo Reference Type

The OPC UA *ReferenceType* IsConnectedTo is used to describe dependencies between objects which are mounted or mechanically linked or connected to each other. The IsConnectedTo reference is symmetric and has no InverseName.

Example for usage in this companion specification: a motor IsConnectedTo to a gear and vice versa.

Table 56 – IsConnectedTo Reference Definition

Attributes	**Value**				
BrowseName	IsConnectedTo				
InverseName					
Symmetric	True				
IsAbstract	False				
Subtype of the HierarchicalReferenceType defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.6 HasSafetyStates (SafetyStatesOf) Reference Type

The OPC UA *ReferenceType* HasSafetyStates is used to describe dependencies between objects to show which (controller) object is responsible for the execution of the safety-functionality. The BrowseName HasSafetyStates and the InverseName SafetyStatesOf describe semantically the hierarchical dependency.

Example for usage in this companion specification: a controller HasSafetyStates and the reference shows to an instance of SafetyStatesType. It is possible that there are two controller in one motion device system

Table 57 – HasSafetyStates Reference Definition

Attributes	**Value**				
BrowseName	HasSafetyStates				
InverseName	SafetyStatesOf				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferenceType defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

9 OPC UA VariableTypes

9.1 EUItemType VariableType Definition

9.1.1 Overview

The *EUItemType* defines the general characteristics of a engineering unit item with a mandatory engineering unit but with optional *EURange* and optional *Definition* properties. All other engineering unit item types derive from it. The *EUItemType* derives from the *DataItemType* and shares the variable model as described in Part 3 and Part 5. The *EUItemType* is used for values with an engineering unit but without a normative range.

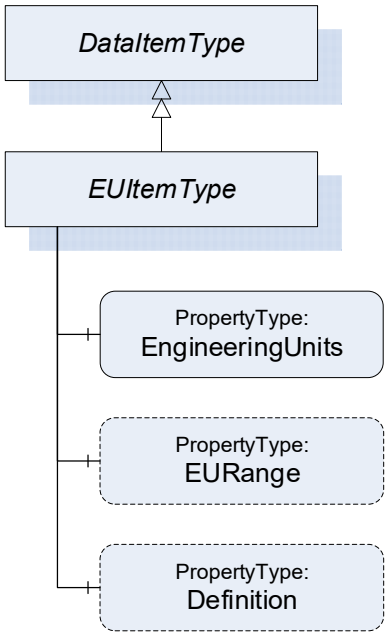

Figure 27 – Overview EUItemType

9.1.2 ObjectType definition

Table 58 – EUItemType Definition

Attribute	Value				
BrowseName	EUItemType				
IsAbstract	False				
DataType	Number				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the DataItemType defined in OPC Unified Architecture Part 8					
HasProperty	Variable	0:EngineeringUnits	EUInformation	PropertyType	Mandatory
HasProperty	Variable	0:EURange	Range	PropertyType	Optional
HasProperty	Variable	0:Definition	Definition	PropertyType	Optional

9.1.3 ObjectType description

9.1.3.1 Variable EngineeringUnits

EngineeringUnits specifies the units for the *DataItem*'s value (e.g., DEGC, hertz, seconds). The *EUInformation* type is specified in 5.6.3 of Part 8 of the OPC UA specification.

9.1.3.2 Variable EURange

The *EURange* property defines the value range likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.

9.1.3.3 Variable Definition

The *Definition* property is a vendor specific, human readable string that specifies how the value of this *EUDataItem* is calculated. *Definition* is non-localized and will often contain additional description for the *EngineeringUnit* e.g. whether the unit voltage is used for DC or AC.

9.2 VectorType VariableType

9.2.1 Overview

This complex *VariableType* is used for information about vectors. Its *DataVariables* reflect its *DataType* having the same semantic defined in 10.2. Additionally, it can define a unit for the vector. The *VariableType* is formally defined in Table 59.

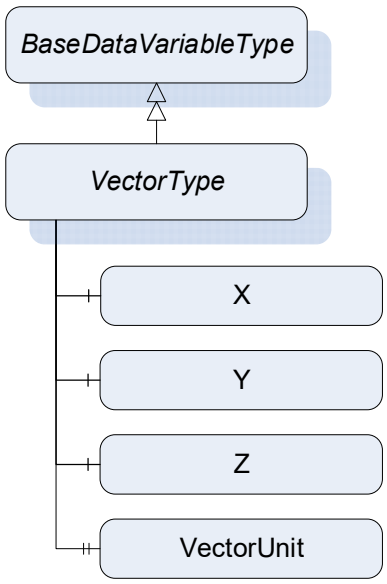

Figure 28 – Overview VectorType

9.2.2 VectorType definition

Table 59 – VectorType Definition

Attribute		Value			
BrowseName		VectorType			
IsAbstract		False			
DataType		Vector			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in OPC Unified Architecture					
HasComponent	Variable	X	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	Y	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	Z	Double	BaseDataVariableType	Mandatory
HasProperty	Variable	0:EngineeringUnits	EUInformation	BaseDataVariableType	Optional

9.2.3 VariableType description

X, *Y* and *Z* reflect the *DataType Vector*. *VectorUnit* defines the unit of the variables *X*, *Y* and *Z*.

9.3 PositionType VariableType

9.3.1 Overview

This complex *VariableType* is used for information about positions. Its *DataVariables* reflect its *DataType* having the same semantic defined in 10.3. Additionally, it can define a unit for the position. The *VariableType* is formally defined in Table 60.

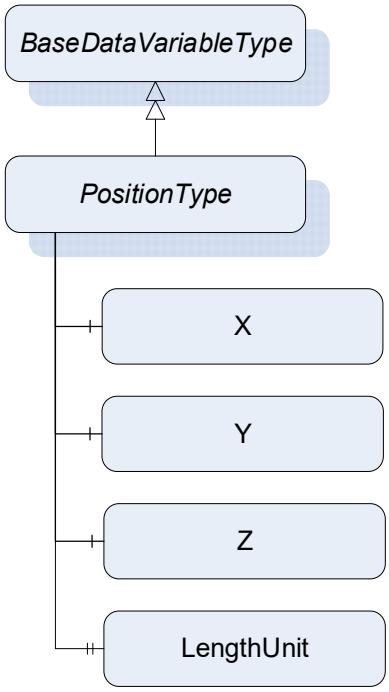

Figure 29 – Overview PositionType

9.3.2 VariableType definition

Table 60 – PositionType Definition

Attribute	Value				
BrowseName	PositionType				
IsAbstract	False				
DataType	Position				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in OPC Unified Architecture					
HasComponent	Variable	X	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	Y	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	Z	Double	BaseDataVariableType	Mandatory
HasProperty	Variable	0:EngineeringUnits	EUInformation	BaseDataVariableType	Optional

9.3.3 VariableType description

X, *Y* and *Z* reflect the *DataType Position*.

LengthUnit defines the unit of the variables *X*, *Y* and *Z*.

9.4 FrameType VariableType

9.4.1 Overview

This complex *VariableType* is used for information about frames. Its *DataVariables* reflect its *DataType* having the same semantic defined in 10.4. Additionally, it provides information about the dynamic behavior of the frame and it can define a unit for the orientation angles. The *VariableType* is formally defined in Table 61.

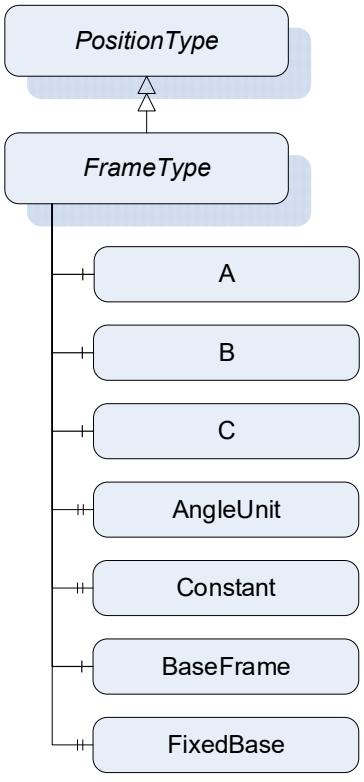

Figure 30 – Overview FrameType

9.4.2 VariableType definition

Table 61 – FrameType Definition

Attribute		Value			
BrowseName		FrameType			
IsAbstract		False			
DataType		Position			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the PositionType defined in OPC Unified Architecture					
HasComponent	Variable	A	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	B	Double	BaseDataVariableType	Mandatory
HasComponent	Variable	C	Double	BaseDataVariableType	Mandatory
HasProperty	Variable	AngleUnit	EUInformation	PropertyType	Optional
HasProperty	Variable	Constant	Boolean	PropertyType	Optional
HasComponent	Variable	BaseFrame	NodeId	BaseDataVariableType	Optional
HasProperty	Variable	FixedBase	Boolean	PropertyType	Optional

9.4.3 VariableType description

A, *B* and *C* reflect the *DataType Frame*.

AngleUnit defines the unit of the variables *A*, *B* and *C*.

Constant is *True* when the *DataVariables* (*X*, *Y*, *Z*, *A*, *B* and *C*) of the frame never change. When these variables may change it is *False*.

BaseFrame is the *NodeId* of the frame's base frame.

FixedBase is True when this frame is always based on the same base frame, i.e. *BaseFrame* never changes.

10 OPC UA DataTypes

10.1 RationalNumber DataType

This *Structured DataType* defines a rational number as a fraction of two integral numbers. Its elements are described in Table 62.

Table 62 – RationalNumber Structure

Name	Type	Description
RationalNumber	structure	
Numerator	Integer	Numerator of the rational number. The sign of the rational number is reflected in this value
Denominator	UInteger	Denominator of the rational number

Its representation in the *AddressSpace* is defined in Table 63.

Table 63 – RationalNumber Definition

Attributes	Value
BrowseName	RationalNumber

10.2 Vector DataType

This *Structured DataType* defines a vector in 3D space. Its elements are described in Table 64.

Table 64 – Vector Structure

Name	Type	Description
Vector	structure	
X	Double	X direction
Y	Double	Y direction
Z	Double	Z direction

Its representation in the *AddressSpace* is defined in Table 65.

Table 65 – Vector Definition

Attributes	Value
BrowseName	Vector

10.3 Position DataType

This *Structured DataType* defines a position in 3D space. Its elements are described in Table 66.

Table 66 – Position Structure

Name	Type	Description
Position	structure	
X	Double	X position
Y	Double	Y position
Z	Double	Z position

Its representation in the *AddressSpace* is defined in Table 67.

Table 67 – Position Definition

Attributes	Value
BrowseName	Position

10.4 Frame DataType

This *Structured DataType* defines a frame in 3D space. It is derived from *Position DataType* and adds orientation information. Further details on frames can be found in C.1.3. Its elements are described in Table 68.

Table 68 – Frame Structure

Name	Type	Description
Frame	Position	
A	Double	Rotation around X (roll)
B	Double	Rotation around Y (pitch)
C	Double	Rotation around Z (yaw)

Its representation in the *AddressSpace* is defined in

Table 69.

Table 69 – Frame TypeDefinition

Attributes	Value
BrowseName	Frame

11 Profiles and Namespaces

11.1 Namespace Metadata

Namespace Metadata are required for any companion standard that specifies an information model (e.g. Objects and Object Types). The metadata provide standardized information about the elements of this namespace. This information is particularly important for aggregating Servers.

Table 70 defines the namespace metadata for this specification. The *Object* is used to provide version information for the namespace and an indication about static *Nodes*. Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. See Part5 for more details.

The information is provided as *Object* of type *NamespaceMetadataType*. This *Object* is a component of the *Namespaces Object* that is part of the *Server Object*. The *NamespaceMetadataType ObjectType* and its *Properties* are defined in Part5.

The version information is also provided as part of the ModelTableEntry in the UANodeSet XML file. The UANodeSet XML schema is defined in Part 6.

Table 70 – NamespaceMetadata Object for this Specification

Attribute	**Value**		
BrowseName	http://opcfoundation.org/UA/<short name>/		
References	**BrowseName**	**DataType**	**Value**
HasProperty	NamespaceUri	String	http://opcfoundation.org/UA/<short name>/
HasProperty	NamespaceVersion	String	x.y
HasProperty	NamespacePublicationDate	DateTime	yyyy-mm-dd
HasProperty	IsNamespaceSubset	Boolean	True or False
HasProperty	StaticNodeIdTypes	IdType[]	
HasProperty	StaticNumericNodeIdRange	NumericRange[]	
HasProperty	StaticStringNodeIdPattern	String	

11.2 Conformance Units and Profiles

This chapter defines the corresponding *Profiles* and *Conformance Units* for the OPC UA Information Model for <title>. *Profiles* are named groupings of *Conformance Units*. *Facets* are *Profiles* that will be combined with other *Profiles* to define the complete functionality of an OPC UA *Server* or *Client.*

11.3 Server Facets

The following tables specify the *Facets* available for *Servers* that implement the <title> Information Model companion specification.

A specification can define multiple facets if not all features are to be implemented by all servers and clients. The name of the facet shall give a hint of the subset. An overall description shall be provided that explains the subset and it potential use. The following table is a template for a facet.

Table 71 defines a facet for the minimum functionality necessary ….

Table 71 – *Template Server Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
CU 1	Supports ….	M
CU 2	Supports ….	M
CU 3	Supports ….	O
Profile		
ComplexType Server Facet (defined in OPC UA Part 7)		M
BaseDevice_Server_Facet (defined in OPC UA Part 100)		M

11.4 Client Facets

The following tables specify the *Facets* available for *Clients* that implement the <title> Information Model companion specification.

A specification can define multiple facets if not all features are to be implemented by all servers and clients. The name of the facet shall give a hint of the subset. An overall description shall be provided that explains the subset and it potential use. The following table is a template for a facet.

Table 72 defines a facet for the minimum functionality necessary for ….

Table 72 – *Template Client Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
CU 1	Uses ….	M
CU 2	Uses ….	M
CU 3	Uses ….	O
Profile		
Method Client Facet (defined in OPC UA Part 7)		M
BaseDevice_Client_Facet (defined in OPC UA Part 100)		M

11.5 Handling of OPC UA Namespaces

Namespaces are used by OPC UA to create unique identifiers across different naming authorities. The *Attributes NodeId* and *BrowseName* are identifiers. A *Node* in the UA *AddressSpace* is unambiguously identified using a *NodeId*. Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*. They are used to build a browse path between two *Nodes* or to define a standard *Property*.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the *EngineeringUnits Property*. All *NodeIds* of *Nodes* not defined in this specification shall not use the standard namespaces.

Table 73 provides a list of mandatory and optional namespaces used in an <title> OPC UA *Server*.

Table 73 – Namespaces used in a <title> Server

NamespaceURI	Description	Use
http://opcfoundation.org/UA/	Namespace for *NodeIds* and *BrowseNames* defined in the OPC UA specification. This namespace shall have namespace index 0.	Mandatory
Local Server URI	Namespace for nodes defined in the local server. This may include types and instances used in an AutoID Device represented by the server. This namespace shall have namespace index 1.	Mandatory
http://opcfoundation.org/UA/DI/	Namespace for *NodeIds* and *BrowseNames* defined in OPC UA Part 100. The namespace index is server specific.	Mandatory
http://opcfoundation.org/UA/<title>/	Namespace for *NodeIds* and *BrowseNames* defined in this specification. The namespace index is server specific.	Mandatory
Vendor specific types and instances	A server may provide vendor-specific types like types derived from *ObjectTypes* defined in this specification or vendor-specific instances of those types in a vendor-specific namespace.	Optional

Table 74 provides a list of namespaces and their index used for *BrowseNames* in this specification. The default namespace of this specification is not listed since all *BrowseNames* without prefix use this default namespace.

Table 74 – Namespaces used in this specification

NamespaceURI	Namespace Index	Example
http://opcfoundation.org/UA/	0	0:EngineeringUnits
http://opcfoundation.org/UA/DI/	2	2:DeviceRevision

Annex A (normative): <Title> Namespace and mappings

A.1 Namespace and identifiers for <Title> Information Model

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this specification. The identifiers are specified in a CSV file with the following syntax:

```
<SymbolName>, <Identifier>, <NodeClass>
```

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is the numeric value for the *NodeId*.

The *BrowsePath* for an *Instance Node* is constructed by appending the *BrowseName* of the instance *Node* to the *BrowseName* for the containing instance or type. An underscore character is used to separate each *BrowseName* in the path. Let's take for example, the <*type*> *ObjectType Node* which has the <*propery*> *Property*. The **Name** for the <*propert*> *InstanceDeclaration* within the <*type*> declaration is: *AutoIdDeviceType_DeviceLocation*.

A NamespaceURI follows the convention: http://opcfoundation.org/UA/<short name>/ where the short name is all caps if an acronym or camel case if words.

Exception if the short name is a trademark. Use trademark casing.

Note that NamespaceURIs are NOT live URLs. Text in the specification should not suggest that they are.

The *NamespaceUri* for all *NodeIds* defined here is http://opcfoundation.org/UA/<short name>/

File Locations

The location of any version dependent files follow this convention:

http://opcfoundation.org/UA/schemas/<short name>/<version>/<file name>

The <short name> is the same as specified in the NamespaceURI;

The <version> is a number with the form #.# or #.##;

The location of the version independent files are the same but with the <version> omitted.

e.g. http://opcfoundation.org/UA/schemas/<short name>/<file name>

File Names

NodeIds: Opc.Ua.<short name>.NodeIds.csv or <short name>.NodeIds.csv

NodeSet: Opc.Ua.<short name>.NodeSet.xml or <short name>.NodeSet.xml;

Any other files should have a prefix that provides context when the file is downloaded in a browser.

All published files must be added to GitHub https://github.com/OPCFoundation/UA-Nodeset

This can be done by creating a mantis issue in the "NodeSets, XSDs and Generated Code" project:

https://opcfoundation-onlineapplications.org/mantis/main_page.php

The files should be attached to the mantis issue.

If the NodeSet was generated with the Opc.Ua.ModelCompiler the design file should be attached as well.

The CSV released with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/<short name>/1.0/NodeIds.csv

NOTE The latest CSV that is compatible with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/<short name>/NodeIds.csv

A computer processible version of the complete Information Model defined in this specification is also provided. It follows the XML Information Model schema syntax defined in Part 6.

The Information Model Schema released with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/<short name>/1.0/Opc.Ua.<short name>.NodeSet2.xml

NOTE The latest Information Model schema that is compatible with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/short name>/Opc.Ua.<short name>.NodeSet2.xml

A.2 Profile URIs for <Title> Information Model

Table 75 defines the Profile URIs for the <title> Information Model companion specification.

Table 75 – Profile URIs

Profile	Profile URI
First facet	http://opcfoundation.org/UA-Profile/External/<short name>/<first facet name>
Second facet	http://opcfoundation.org/UA-Profile/External/<short name>/<second facet name>
…	

A.3 Profile URIs for <Title> Information Model

Annex B (informative): examples

B.1 Examples of motion device systems, motion devices, axes and power trains

This chapter describes examples for motion device systems, motion devices, axes and power trains.

B.1.1 Example for motion device systems

Typically a motion device system consists of at least one manipulator and one control unit. Manipulators shown in Figure 31, Figure 32, x3, x4, x5, x6 and x7 normally have only one control unit.

Figure x8 shows an example with four motion devices which can be controlled by one control unit.

The motion device system illustrated in Figure x9 consists of three motion devices and may have one or more control units regarding the motion devices. When a safety PLC is integrated in this motion device system, it can be described as an own instance of a *ControllerType*. This Instance would have no Reference to an instance of a motion device, because the safety PLC doesn´t control a manipulator. It could however have a Reference to the instantiated *SafetyStates*.

B.1.2 Examples for motion devices and controllers in a motion device system

The motion devices shown in Figure x8 are typically controlled by one controller unit. Each motion device *IsControlledBy* the same controller.

The system illustrated in Figure x9 may have two control units. For example one controller *Controls* the both articulated robots and the mobile platform *IsControlledBy the other controller*.

B.1.3 Examples for motion devices

A motion device can be any manipulator e.g. a robot, a linear unit or a turn table. For each motion device which has an own type plate an instance of a *MotionDeviceType* shall be created.

The kind of motion device shall be described with the *Property DeviceClass* of the *ParameterSet* of the *MotionDeviceType* by the *MotionDeviceClassEnumeration,* which is based on definitions of ISO 8373:2012.

The Figures Figure 31 and Figure 32 show examples of cartesian manipulators.

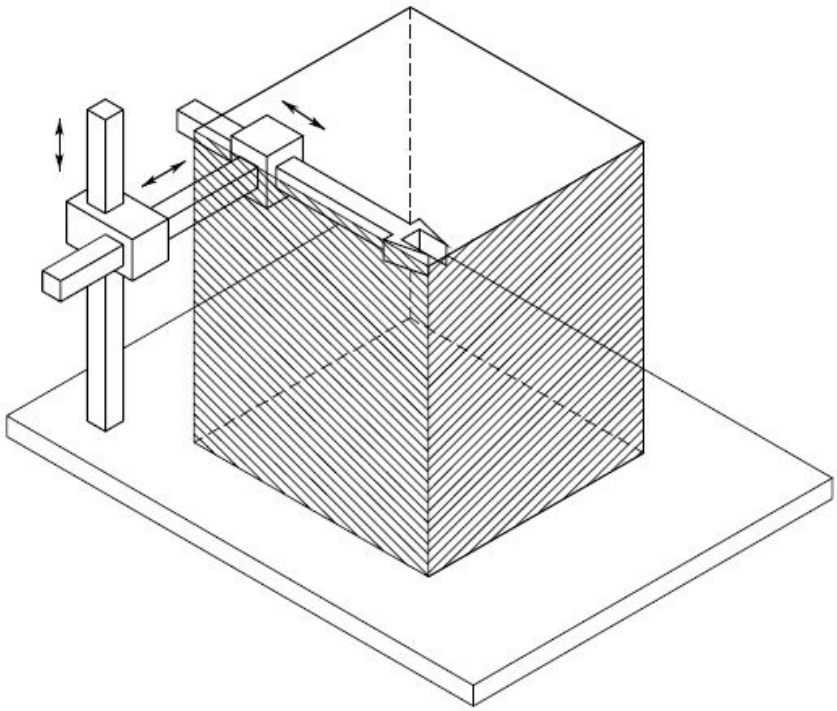

Figure 31 – Cartesian manipulator

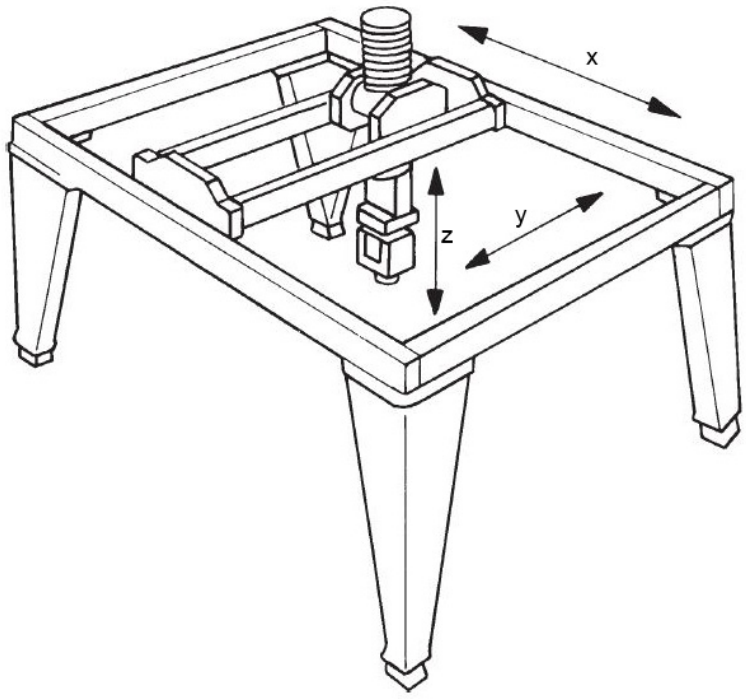

Figure 32 – Portal manipulator

Figure 33 shows an example of a parallel manipulator. So called delta robots, as shown in Figure 34, are also parallel manipulators.

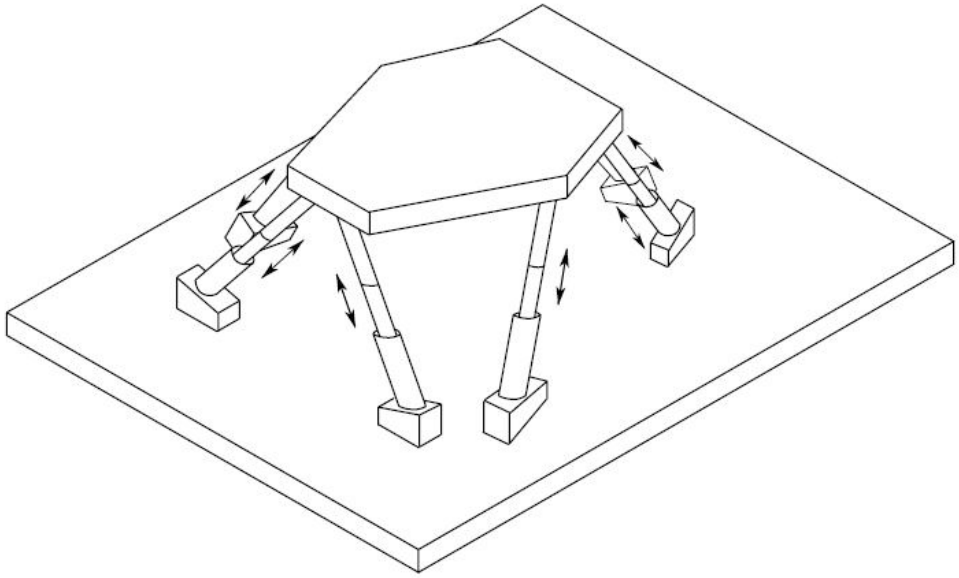

Figure 33 – Stewart platform or Hexapod

Figure 34 – Delta robot

Figure 35 shows an abstract example of a scara robot.

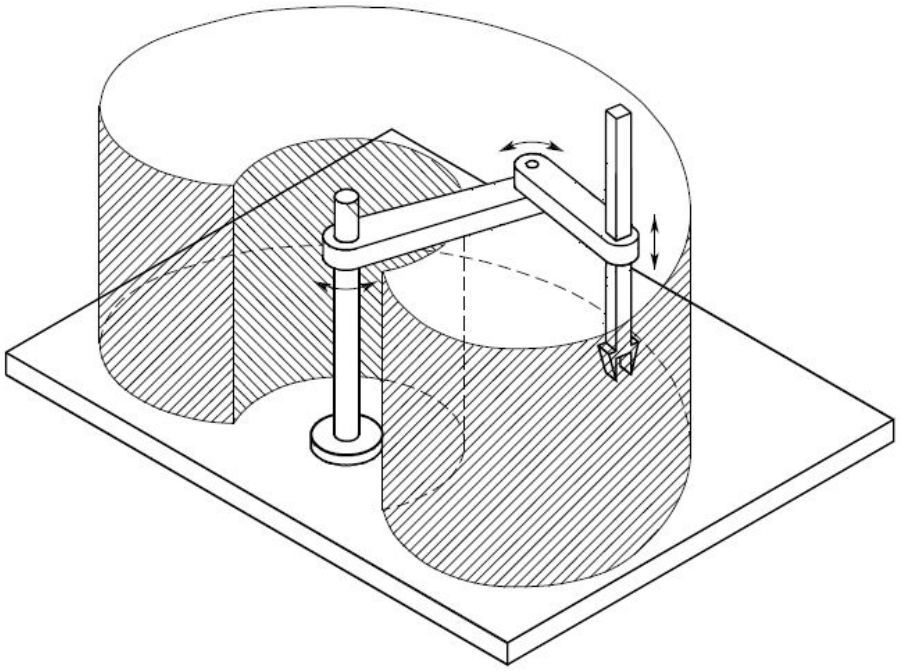

Figure 35 – Scara robot

A typical example of an articulated robot is shown in Figure 36.

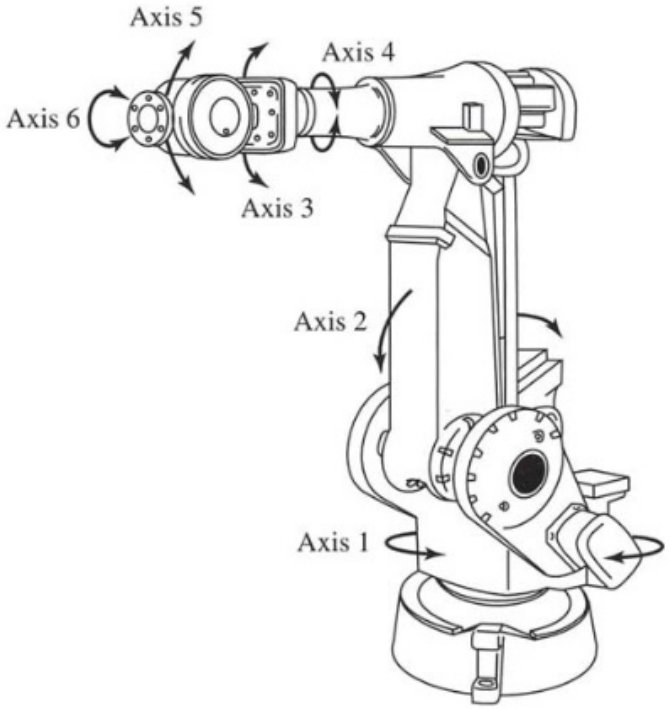

Figure 36 – Articulated robot

Another example of an articulated robot is a so called humanoid robot as Figure 37 schematically shows.

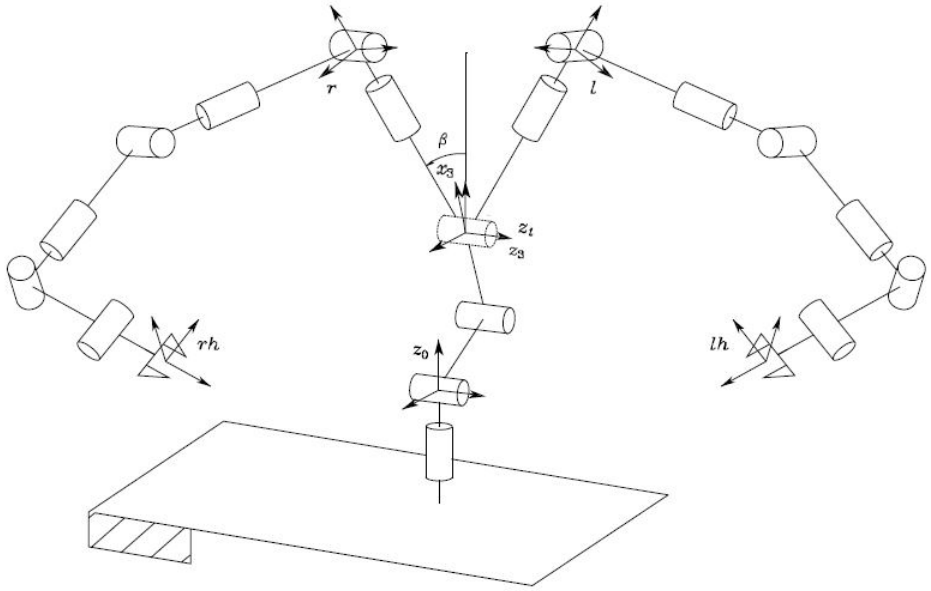

Figure 37 – Schematic of a humanoid robot

B.1.4 Examples of combinations of motion devices in a motion device system

Figure 38 shows four motion devices integrated in one motion device system: an articulated robot on a linear unit with two turntables.

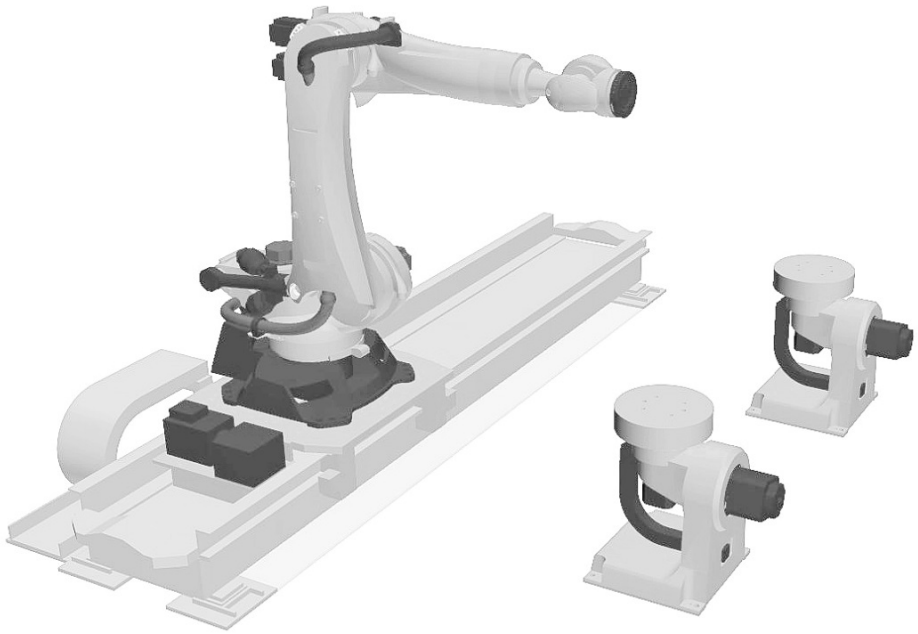

Figure 38 – Motion device system 1

Figure 39 shows three motion devices in one motion device system: two articulated robots on a mobile platform.

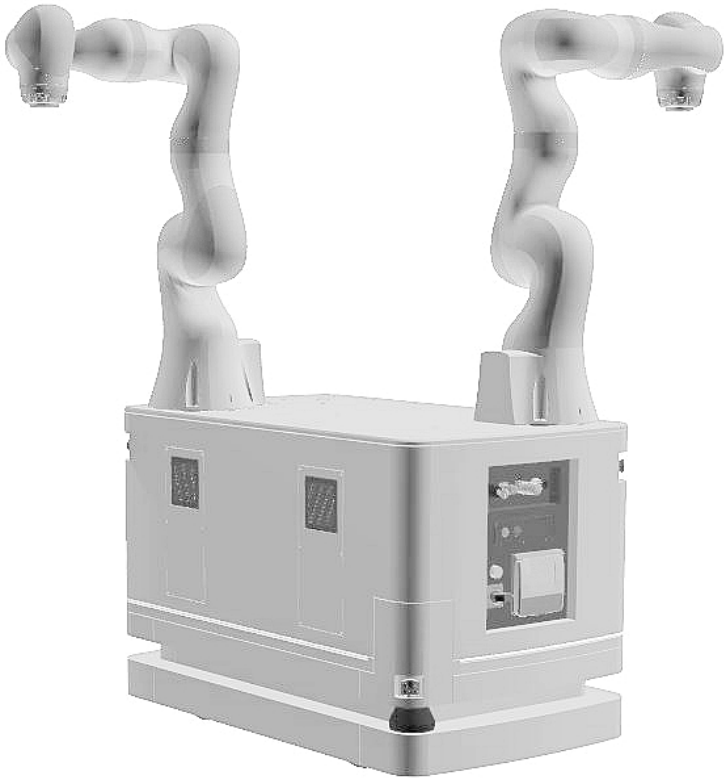

Figure 39 – Motion device system 2

B.1.5 Axes and power trains

An axis of a motion device is the mechanical joint of a manipulator that performs a linear or a rotational movement.

Power trains, consisting of gears, motors and drives, are responsible for the movement of axes. Drives can be integrated in the manipulator or inside a controller cabinet. *References* describe the relationships between the components of the power train.

Figure 40 shows two possibilities for a realization of a linear two-dimensional motion device. While in the left figure there is a 1:1 relation between power train and mechanical axis in the right figure power train 1 and power train 2 have effect on the movement of axis 1 and on axis 2. An additional load is located on the mechanical axis 2 but has effect on both power trains.

References describe the relationships between the movement of axes and the power trains that initiate the movement.

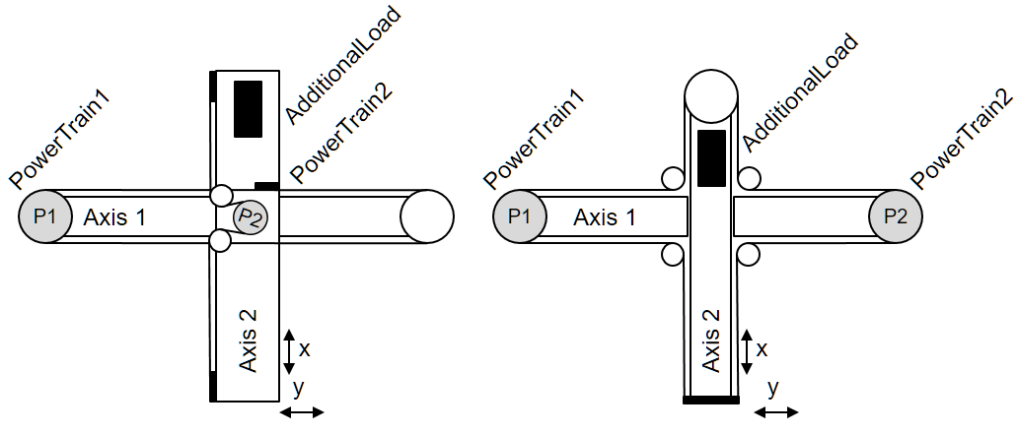

Figure 40 – Axis and power train coupling

B.1.6 Virtual Axes

If there is the need to show information about virtual axes, which are not actively run by a power train, then these axes shall be provided, but they don´t have *References* to a power train. An example for a virtual axis is, when a robot control calculates the movement of an external axis in accordance to the robot movement, e.g. for a servo welding gun mounted at the robot flange, but doesn´t control actively the movement of this axis with an internal power train.

Another example for a virtual axis can be found in a delta robot. When the fourth axis is driven through a telescope shaft and cardan joints, then the length of the telescope shaft is depending on the positions of axes 1, 2 and 3. This length can be seen as a virtual axis, as it has constraints similar to a real axis, e.g. position limits. But it is not possible to actively move this axis.

B.1.7 Examples for axes and power trains

The Figure 31 and Figure 32 show different versions of Cartesian robots. Figure 31 shows a three axis robot which has one dedicated power train for each axis: An axis *IsMovedBy* only one power train and so an axis only *Requires* one dedicated power train. One motor of a power train *IsDrivenBy* a drive and *IsConnectedTo* a gear.

A variant of the robot shown in Figure x2 could have a master-slave drive setup for the movement of the x-axis. The y-axis and the z-axis are moved each by one motor. In this case the Powertrain of the x-axis contains two motors, drives and gears.

The *Reference Requires* (*IsRequiredBy*) is currently not used in the type definitions of Chapter 7.

B.1.8 Examples for the use of References regarding axes and power trains

For the right Figure 40 the *References* between axes and power trains would be:

i. Power train 1 *Moves* axis 1 and axis 2
ii. Power train 2 *Moves* axis 1 and axis 2
iii. Axis 1 *IsMovedBy* power train 1 and power train 2
iv. Axis 2 *IsMovedBy* power train 1 and power train 2
v. Axis 1 *Requires* power train 1 and power train 2
vi. Axis 2 *Requires* power train 1 and power train 2
vii. Power Train 1 *IsRequiredBy* axis 1 and axis 2
viii. Power Train 2 *IsRequiredBy* axis 1 and axis 2

The typical six-axis industrial robot shown in Figure 36 normally has 6 power trains for the movement of the 6 axes. Due to the robot hand design, various power trains initiate internal compensation movements. When

only the motor of power train 4 is rotating then axis 4, axis 5 and 6 are moving. When only axis 4 should be moved and axis 5 and 6 should stand still then power trains 5 and 6 must compensate the movement of these axes. Thus a movement of only axis 4 requires rotation of the motors of the power trains 4, 5 and 6.

There are two views regarding the *References*: The idea is to use the semantic of the *Reference* to describe the object in reference to the subject.

A power train *Moves* an axis means that if a motor of this power train moves there will be an effect of the position of the axis.

ix. Power train 1 *Moves* axis 1
x. Power train 2 *Moves* axis 2
xi. Power train 3 *Moves* axis 3
xii. Power train 4 *Moves* axis 4, axis 5 and axis 6
xiii. Power train 5 *Moves* axis 5 and axis 6
xiv. Power train 6 *Moves* axis 6

Description regarding iv.: when only the motor of power train 4 is moving there is an effect on the position of axis 4, axis 5 and axis 6.

An axis *IsMovedBy* a power trains means, that actions of these power trains have an influence on the axis position.

i. Axis 1 *IsMovedBy* power train 1
ii. Axis 2 *IsMovedBy* power train 2
iii. Axis 3 *IsMovedBy* power train 3
iv. Axis 4 *IsMovedBy* power train 4
v. Axis 5 *IsMovedBy* power train 5 and power train 4
vi. Axis 6 *IsMovedBy* power train 6, power train 5 and power train 4

Description regarding vi.: Axis 6 movement is depending on movement from power train 6, power train 5 and power train 4.

An axis *Requires* the movement of a motor of a power train to position but also other power trains might be involved by this movement to compensation movements of affected axes.

i. Axis 1 *Requires* power train 1
ii. Axis 2 *Requires* power train 2
iii. Axis 3 *Requires* power train 3
iv. Axis 4 *Requires* power train 4, power train 5 and power train 6
v. Axis 5 *Requires* power train 5 and power train 6
vi. Axis 6 *Requires* power train 6

Description regarding iv.: when only axis 4 should be moved, there are compensation movements commanded by power train 5 and power train 6 to ensure a standstill of axis 5 and axis 6.

A power train IsRequiredBy axes means that this power train is active when only the referenced axis should be moved and all other axis should stand still.

i. Power train 1 *IsRequiredBy* axis 1
ii. Power train 2 *IsRequiredBy* axis 2
iii. Power train 3 *IsRequiredBy* axis 3
iv. Power train 4 *IsRequiredBy* axis 4
v. Power train 5 *IsRequiredBy* axis 4 and axis 5
vi. Power train 6 *IsRequiredBy* axis 4, axis 5 and axis 6

Description regarding vi: power train 6 is involved in positioning of axis 4, axis 5 and axis 6.

Annex C (informative): Outlook and concepts

C.1 Outlook and concepts

C.1.1 State machine

Currently a state machine to control motion device systems is under development.

The approach is to use states and transitions to monitor process steps in a client application. One of the main use cases is to gather data for analytic purposes, e.g. the duration of a running process, idle times or the duration of a production interruption caused by an error.

With the possibility to load and unload task programs the state machine is also a first step to control a robot remotely via OPC UA. In this context a task program either describes a full production sequence or sub-sequences of production steps.

The state machine supports transitions and useful remote-functions to cover the following use cases and proves the concept of the state machine:

- List available task programs in a controller
- Upload of task programs to a controller
- Download of a task program from the controller
- Loading a task program in a task control
- Unloading of a task program in a task control
- Overloading of a task program in a task control
- Positioning of the manipulator in the context of the loaded task program (if supported by the system)
- Starting of a task program
- Stopping of a task program
- Resetting of a task program
- Confirmation of system messages (if supported by the system)
- Switching drives on and off

Figure 41 shows the concept of the state machine. Transitions are colored different to make them distinguishable by their cause.

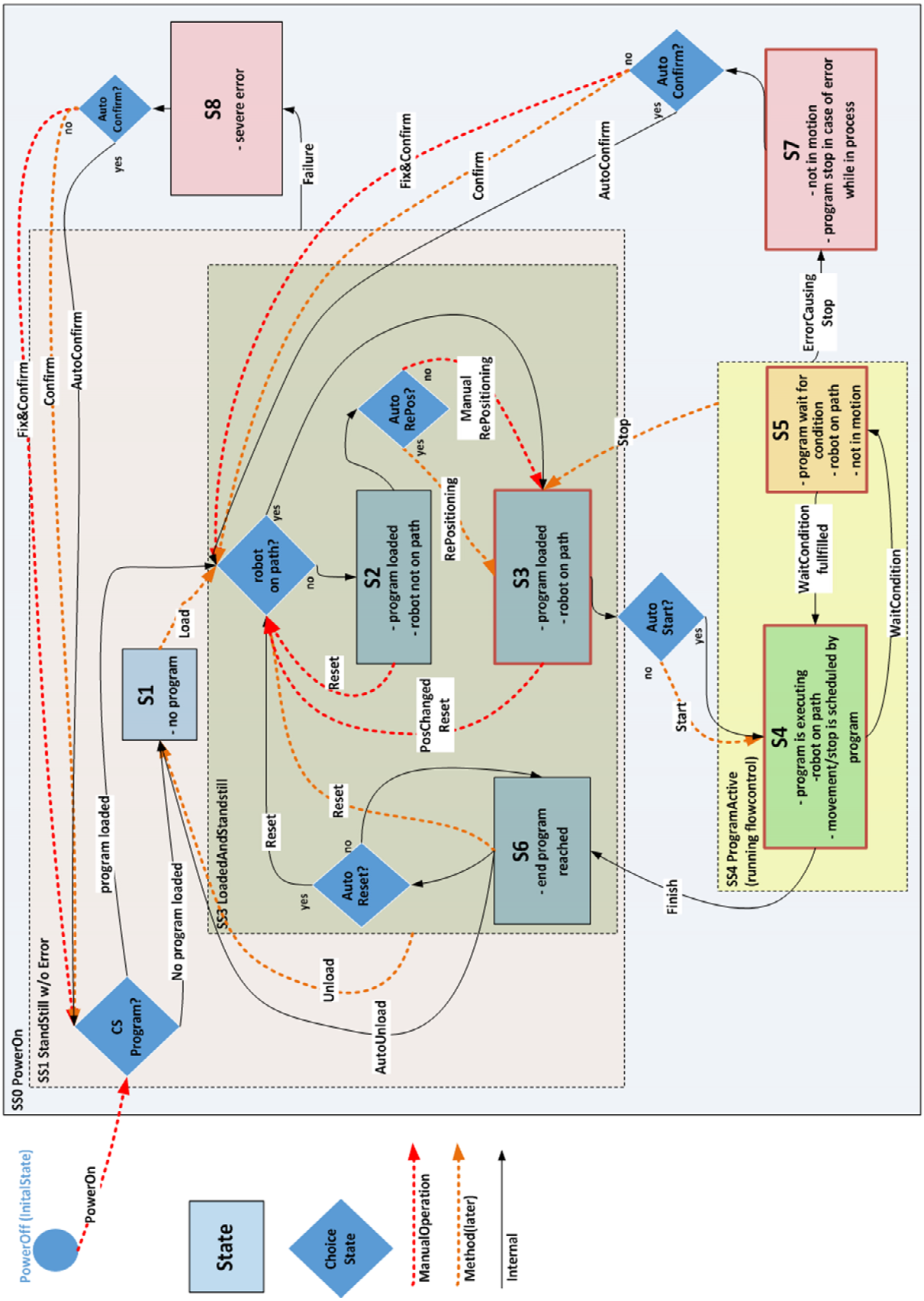

Figure 41 – *State machine*

C.1.2 Geometry Reference

Currently a layout description of the objects of a motion device system is under development.

This layout description, currently called geometrical reference, will provide a spatial overview of all objects which have a geometrical relevance regarding the production process for monitoring aspects. In further steps to control a motion device system with geometrical commands remotely (via OPC UA) it is necessary to provide the actual spatial status of all relevant objects of the motion device system.

Dynamic aspects of a production cell are supported to ensure changes of geometrical references to enable e.g. a tool change at a robot flange. Therefor a list which organizes all objects with geometrical context has to be instantiated. In case of appearing or disappearing geometrical objects this list can be adapted and a provided mechanism enables a client to update the geometry reference information.

This generic approach to describe geometrical relationships of objects is based on the standard mathematic of frames, coordinate systems and transformations described in chapter Annex C.1.3.

The idea of the concept is that each object of the motion device system, which has a geometrical relationship regarding a cell or a process, describes its own dimension and on which coordinate system it is based on. Therefor an offset frame which has a reference to a coordinate system where it is based on and all interesting frames, e.g. a mounting point where a tool or another motion device can be attached will be provided for each object. The references between the frames enable a client application to build up a view of all chained frames inside the motion device system.

Based on this concept process specific frames e.g. from the tool center point to a workpiece base or the TCP contouring deviation can be extended.

Figure 42 shows an example of a production cell with a robot on a linear unit and the frames inclusive their references to coordinate systems. The blue arrows show the displacement of a frame. The green ones show the references to their base frame.

In this example frame 1 describes where inside a cell the linear unit (MD 2) is placed. Frame 6 shows the offset frame to the carriage of the linear unit. Frame 7 is the dynamic frame to the mounting point at the carriage where the robot is attached. In the example the offset frame of the robot (MD 1) is zero. Frame 3 shows to the mounting point (flange) of the robot. The tool is attached without offset though frame 5 shows the TCP. The reference 5 points to the coordinate system where the tool is mounted on and reference 3 points to the coordinate system for the robot flange frame. The reference 3a points to the zero offset frame of the robot and reference 7 points to the coordinate system of the carriage. The reference 6 points to the base coordinate system of the linear unit and reference 1 points to the cell world.

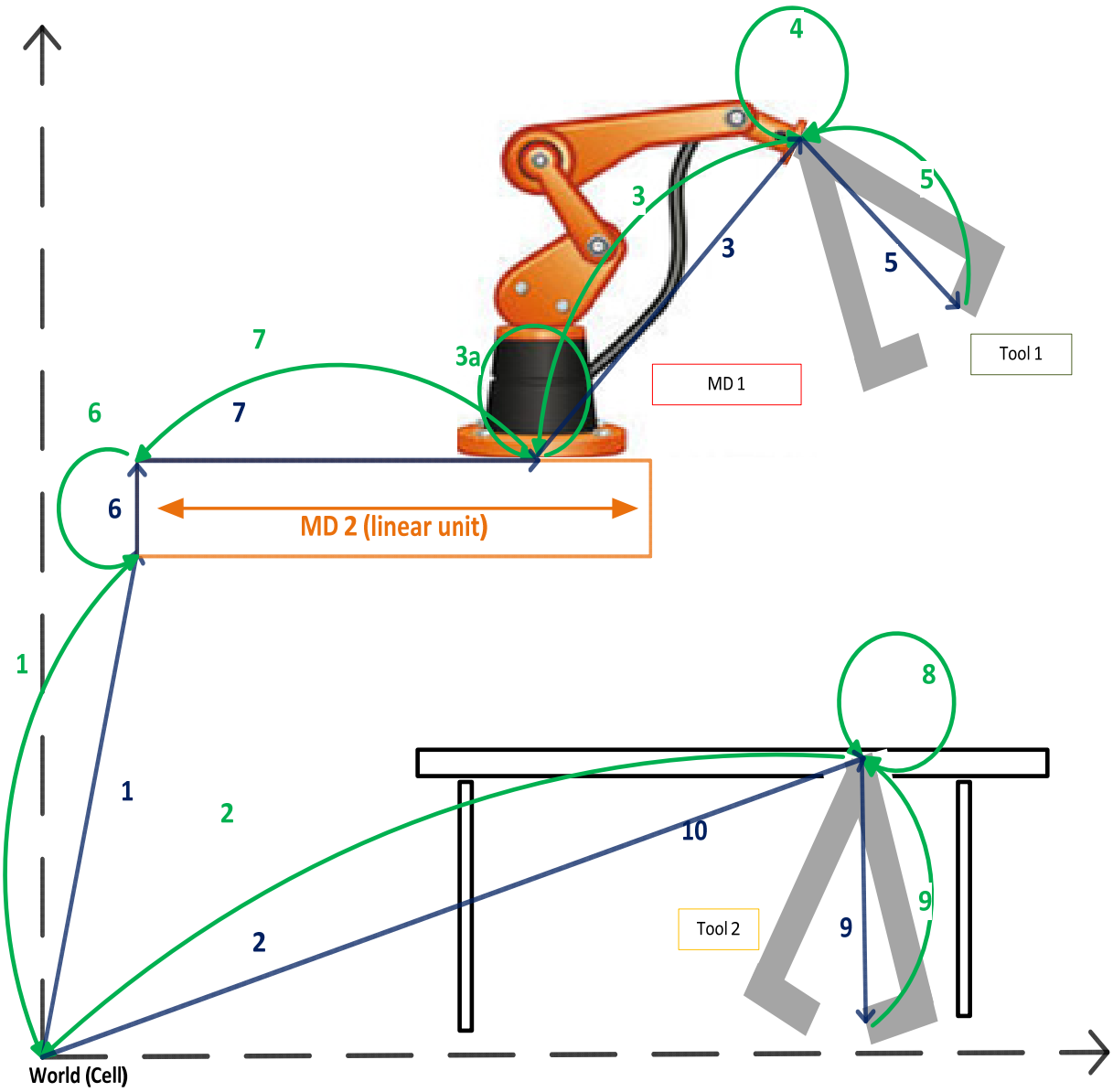

Figure 42 – Example of layout description

Figure 43 shows an abstract view of instances of a OPC UA server for the example shown in Table 42:

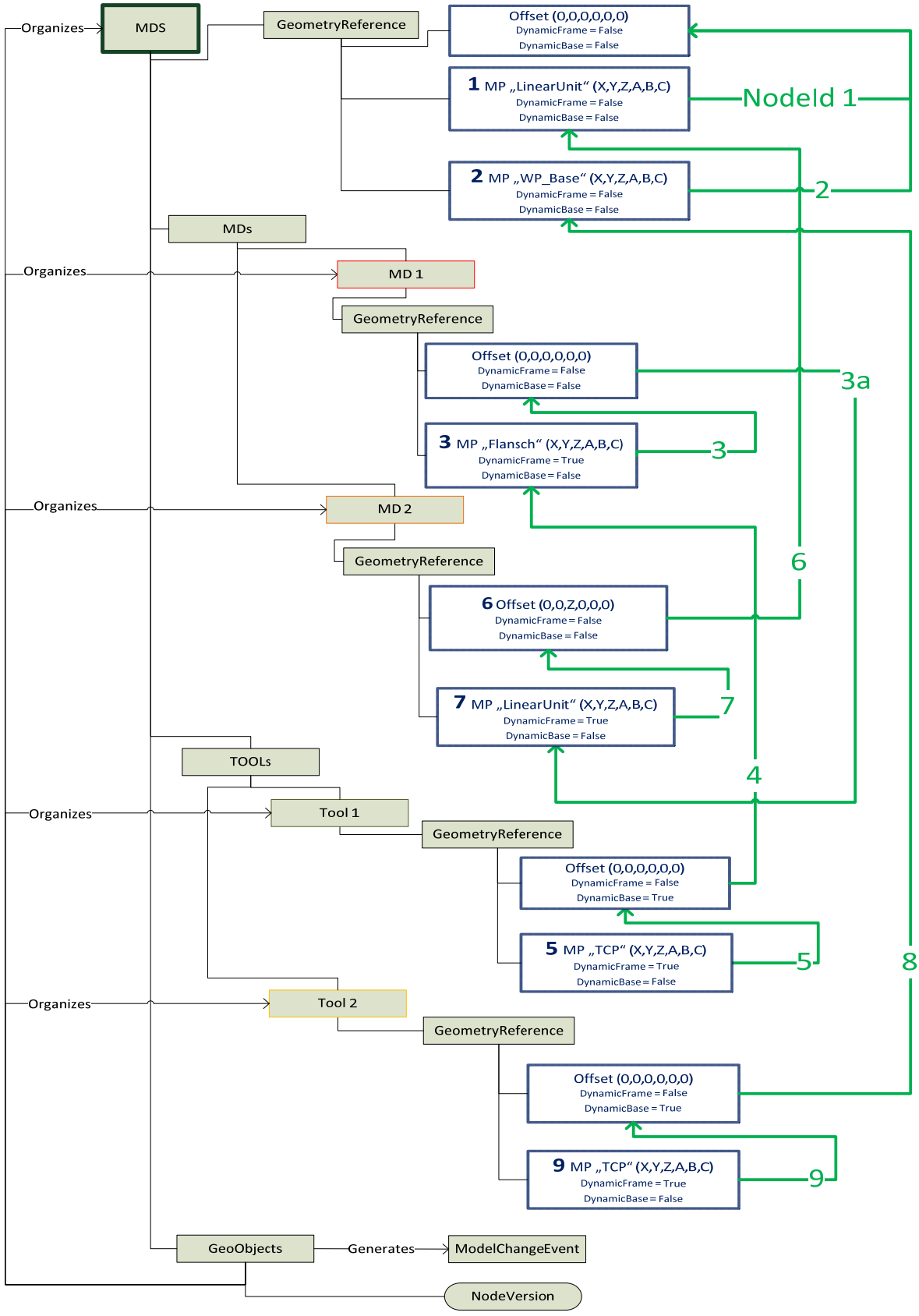

Figure 43 – Abstracted server view of the geometry reference

C.1.3 Geometrical representation of objects reference

The position and orientation of objects in space is a general requirement that will be needed also by other companion specifications. Thus we try to establish a general description.

The geometrical setup of a robotic system can be very diverse. It is not possible to specify all possible setups in the standard. Instead, the standard provides the framework and gives examples for how to use.

Frames

A frame describes the translation and rotation of an object relative to another frame (the base frame). Figure 44 shows a simple frame chain to clarify the notation. The frame A is the base of frames B and C. The frames B and C are based on frame A. Frame C is the base of frames D and E. The frames D and E are based on frame C.

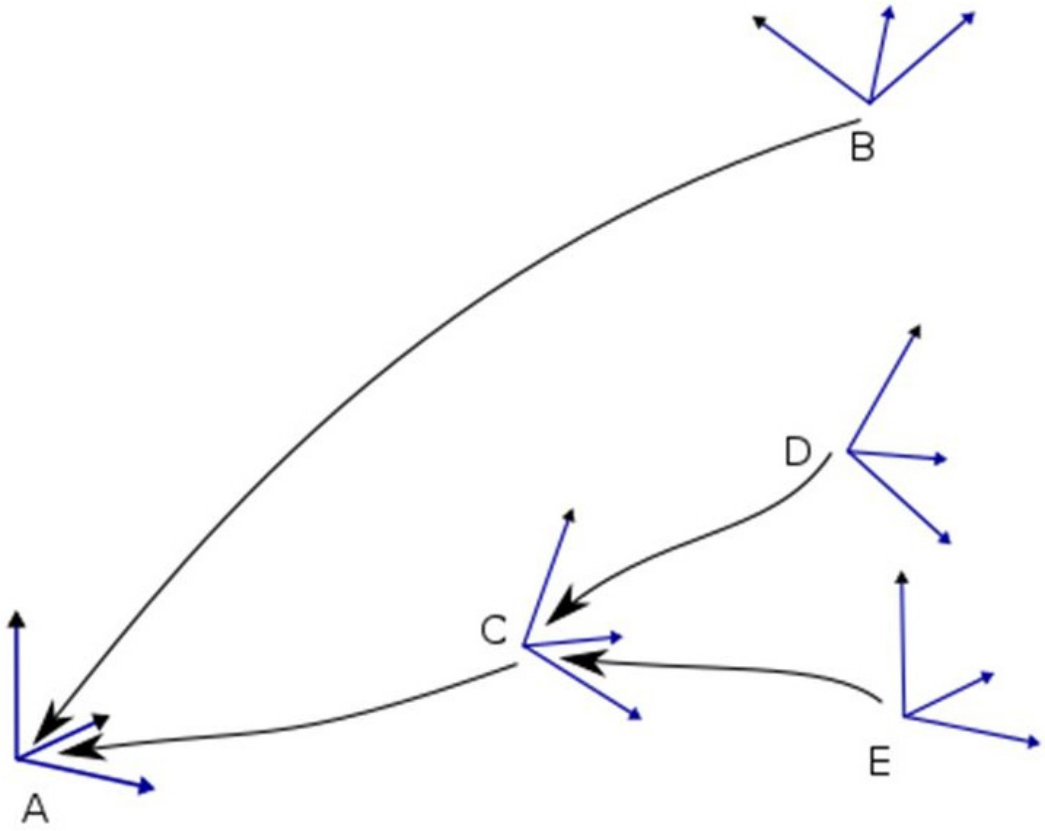

Figure 44 – Simple frame chain

As shown in Figure 44 an arrow points from a frame to its base frame to visualize the frame chains. This arrow can be read as “The coordinates of frame B are specified in (or relative to) Frame A”. In addition to the coordinates of frame B you also need to know its base frame A.

The coordinates of a frame are represented by the three values P = (X, Y, Z) for the position and the three values O = (A, B, C) for the orientation.

P is the translation relative to the base frame. When the translation of a frame is 0 then the origins of the frame and its base frame coincide.

O is the orientation of the frame in yaw, pitch and roll notation (see also ISO 9787:2013 “Robots and robotic devices — Coordinate systems and motion nomenclatures” or Wikipedia article about euler angles).

We use the notation of pre-multiplying rotation-matrices and column vectors.

Figure 45 shows the three elementary rotations A, B and C as specified in ISO 9787:2013.

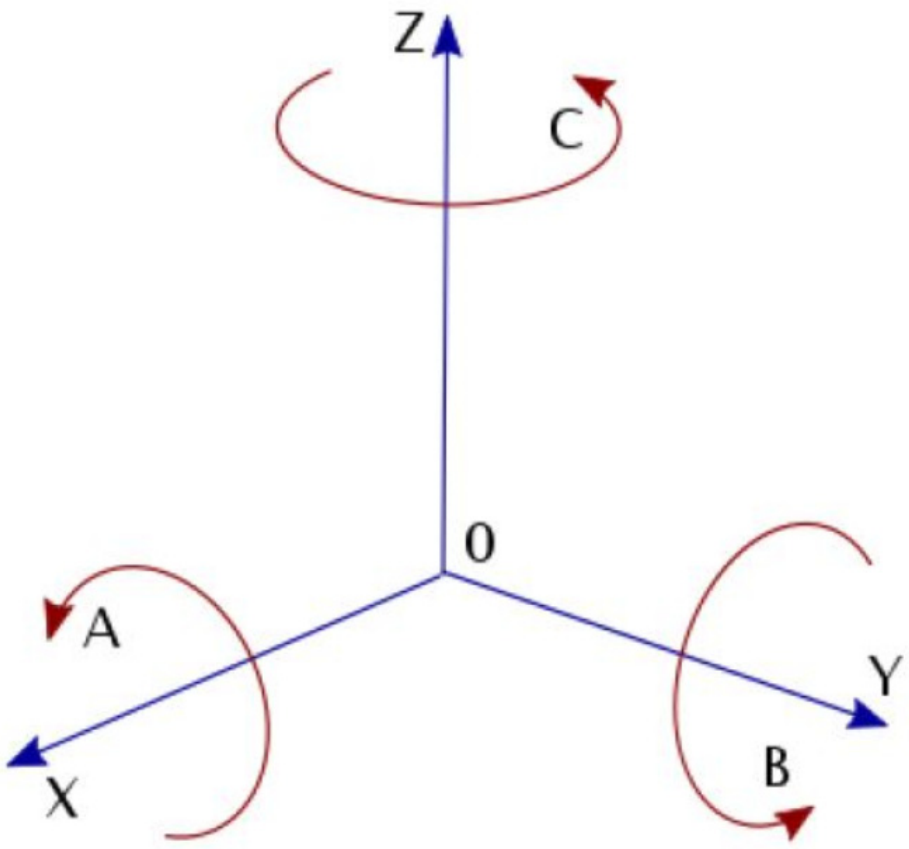

Figure 45 – Rotations

The elementary rotations correspond to the following rotation matrices:

A = roll = rotation about X axis

$$R_X(A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(A) & -\sin(A) \\ 0 & \sin(A) & \cos(A) \end{bmatrix}$$

B = pitch = rotation about Y axis

$$R_Y(B) = \begin{bmatrix} \cos(B) & 0 & \sin(B) \\ 0 & 1 & 0 \\ -\sin(B) & 0 & \cos(B) \end{bmatrix}$$

C = yaw = rotation about Z axis

$$R_Z(C) = \begin{bmatrix} \cos(C) & -\sin(C) & 0 \\ \sin(C) & \cos(C) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ISO 9787:2013 does not define the order of the rotations or the whether to use intrinsic or extrinsic rotations. Thus we refer to the Wikipedia article here. It says that the intrinsic rotations z-y'-x″ are known as yaw, pitch and roll, giving the transformation matrix

$$R(A,B,C) = R_Z(C)R_Y(B)R_X(A) =$$

$$\begin{bmatrix} \cos(B)\cos(C) & \sin(A)\sin(B)\cos(C) - \cos(A)\sin(C) & \sin(A)\sin(C) + \cos(A)\sin(B)\cos(C) \\ \cos(B)\sin(C) & \sin(A)\sin(B)\sin(C) + \cos(A)\cos(C) & \cos(A)\sin(B)\sin(C) - \sin(A)\cos(C) \\ -\sin(B) & \sin(A)\cos(B) & \cos(A)\cos(B) \end{bmatrix}$$

Figure 46 shows the three consecutive rotations:

1) Rotation about the Z axis (blue rotation; blue frame -> green frame)
2) Rotation about the new Y axis (green rotation; green frame -> red frame)
3) Rotation about the new X axis (red rotation; red frame -> black frame)

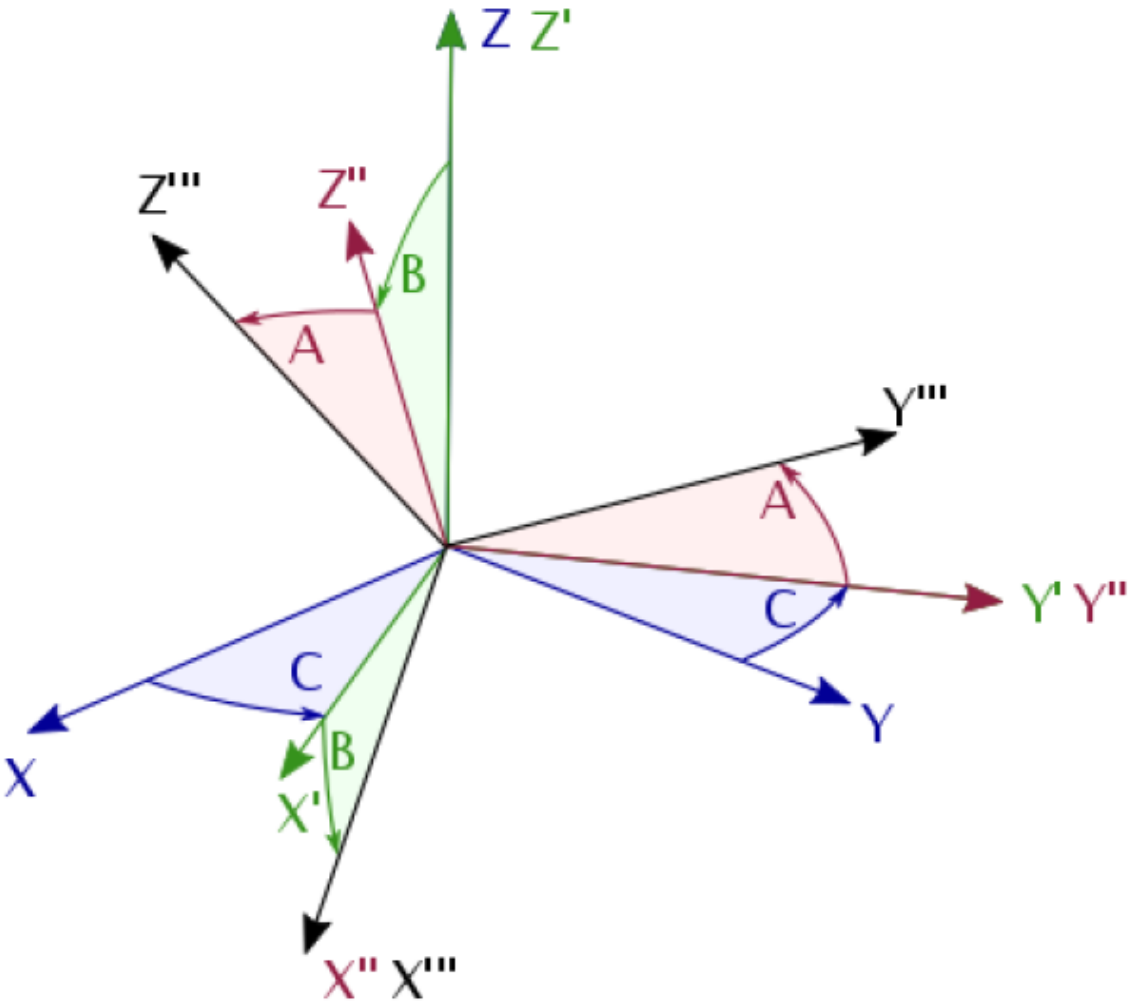

Figure 46 – Rotations of a frame

The extrinsic rotations x-y-z about the axis of the original fixed coordinate system result in the same transformation matrix. This is

1) Rotation about the X axis
2) Rotation about the original Y axis
3) Rotation about the original Z axis

Frames can be constant (e.g. the robot base) or dynamically changing (e.g. the robot flange).