OPC 40010-1

OPC UA for Robotics

Part 1: Vertical Integration

Release 1.00

2019-07

OPC 40010-1 (Edition 1.0, 2019-07) is identical with VDMA 40010-1:2019-07

OPC UA Companion-Specification

VDMA 40010-1

ICS 25.040.30

OPC UA Companion Specification for Robotics (OPC Robotics) – Part 1: Vertical integration

OPC UA Companion Specification for Robotics (OPC Robotics) – Teil 1: Vertikale Integration

Document comprises 80 pages

VDMA

Contents

Figures

Tables

OPC FOUNDATION, VDMA

AGREEMENT OF USE

Foreword

This specification was created by a joint working group of the OPC Foundation and VDMA.

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), 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 – Forschungsinstitut des Freistaats Bayern
– 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

1 Scope

This document specifies 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.

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)
- OPC 10000-3, *OPC Unified Architecture - Part 3: Address Space Model*

  http://www.opcfoundation.org/UA/Part3/
- OPC 10000-4, *OPC Unified Architecture - Part 4: Services*

  http://www.opcfoundation.org/UA/Part4/
- OPC 10000-5, *OPC Unified Architecture - Part 5: Information Model*

  http://www.opcfoundation.org/UA/Part5/
- OPC 10000-8, *OPC Unified Architecture - Part 8: Data Access*

  http://www.opcfoundation.org/UA/Part8/
- OPC 10000-100, *OPC Unified Architecture - Part 100: Devices*

  http://www.opcfoundation.org/UA/Part100/
- OPC 10001-1, *OPC Unified Architecture V1.04 - Amendment 1: AnalogItem Types*
- OPC 10001-11, *OPC Unified Architecture V1.04 - Amendment 11: SpatialTypes*

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 Terms

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	Operational 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 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).
Industrial Robot System	system comprising industrial robot, end effectors and any machinery, equipment, devices, external auxiliary axes or sensors supporting the robot performing its task (ISO 8373)
Joint	See Axis definition.
Manipulator	Machine in which the mechanism usually consists of a series of segments, jointed or sliding relative to one another, for the purpose of grasping and/or moving objects (pieces or tools) usually in several degrees of freedom (ISO 8373)
Manual mode	Control state that allows for the direct control by an operator (ISO 10218).
Motion device	A motion device has as least one axis and is a multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks. Examples are an industrial robot, positioner or mobile platform.
Motion device system	The whole system in which one or more motion devices and one or more controllers are integrated, e.g. a robot system.
Operating mode	State of the robot control system (ISO 8373), i.e. Controller
Operational mode	ISO 10218-1:2011 Ch.5.7 Operational Modes
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 switch gears, fuses, transformers, converters, drives, motors, encoders and gears to convert power to motion of one or more 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).
Safe state	A defined state of the robot which is free of hazards
Safety function	A safety rated function which will signal the controller to bring motion devices to a safe state, e.g. emergency stop, protective stop
Safety states	Set of safety functions and states which are related to a motion device system.
Software	Runtime software or firmware of the controller. In ISO 8373, this is called control program, and is defined like this: Inherent set of control instructions which defines the capabilities, actions and responses of a robot or robot system NOTE This type of program is usually generated before installation and can only be modified thereafter by the manufacturer.
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. From ISO 8373: Set of instructions for motion and auxiliary functions that define the specific intended task of the robot or robot system NOTE 1 This type of program is usually generated after the installation of the robot and can be modified by a trained person under defined conditions. NOTE 2 An application is a general area of work; a task is specific within the application.
Tool center 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 consist 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.

Annex B.1 contains examples of the described terms.

3.3 Abbreviations

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
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
OPC UA DI	OPC Unified Architecture for Devices (DI) OPC Unified Architecture - Part 100 – Devices
PLC	Programmable logic controller
PMS	Preventive Maintenance System
TCP	Tool center point
TCP/IP	Transmission Control Protocol/Internet Protocol
TCS	Tool Coordinate System
UPS	Uninterruptible Power Supply
URL	Uniform resource locator
URI	A uniform resource identifier (URI) is a strings of characters used to identify names or resources on the Internet. The URI describes the mechanism used to access resources, the computers on which resources are housed and the names of the resources on each computer.
VDMA	The Mechanical Engineering Industry Association (VDMA) represents more than 3,200 member companies in the SME-dominated mechanical and systems engineering industry in Germany and Europe.

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 10000-3). 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 10000-3) and the *ArrayDimensions* is set to null or is omitted. Examples are given 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 60 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 10000-3. 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 Figure 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 10000-3.

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 10000-3.

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 10000-3.

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 10000-3.

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:

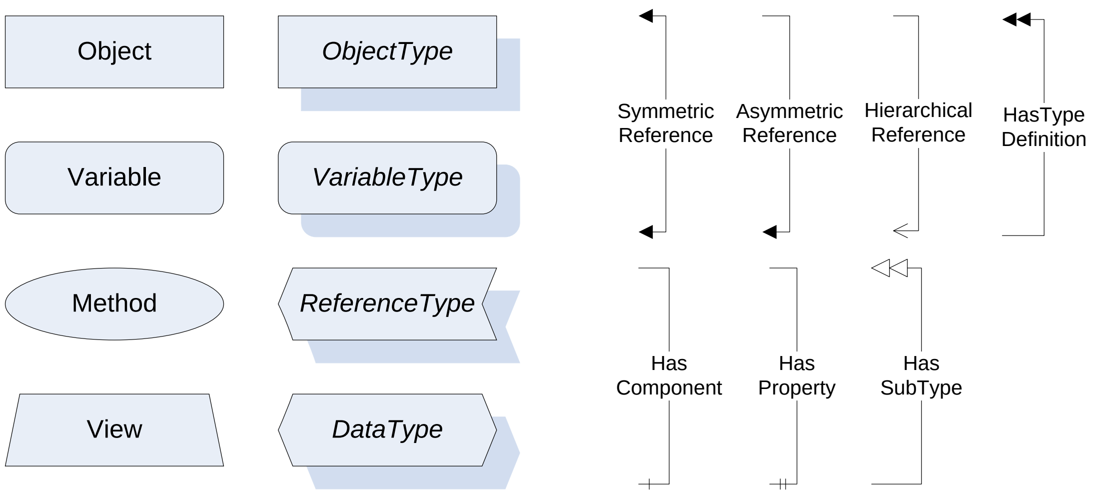


Figure 1 – OPC UA standard definitions

Additional definitions:

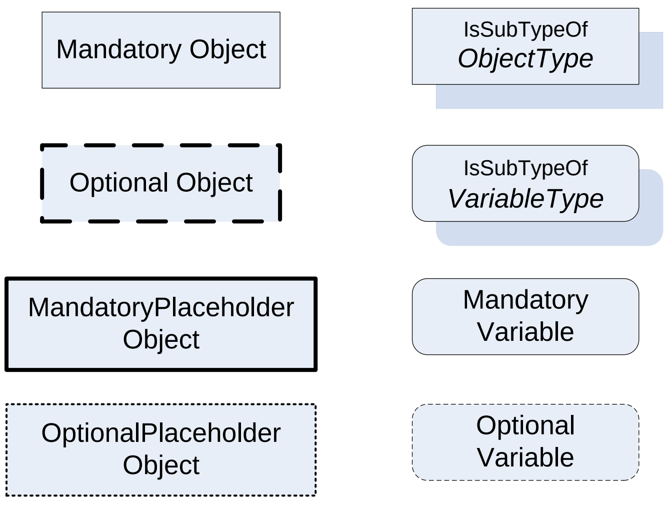


Figure 2 – OPC UA and additional definitions

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 Robotics and OPC UA

4.1 Introduction to OPC Robotics

The OPC 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
– 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 10000-2).
– 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 example. For a more complete overview see OPC 10000-1.

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 10000-4) 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 historical 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.

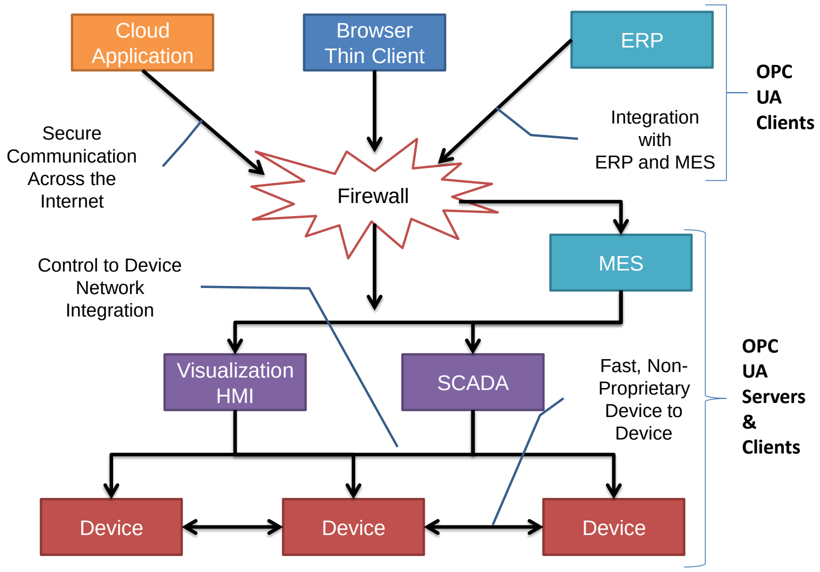


Figure 3 – The Scope of OPC UA within an Enterprise

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 *NodeId* and non-localized name called *BrowseName*. An *Object* representing a 'Reservation' is shown in Figure 4.

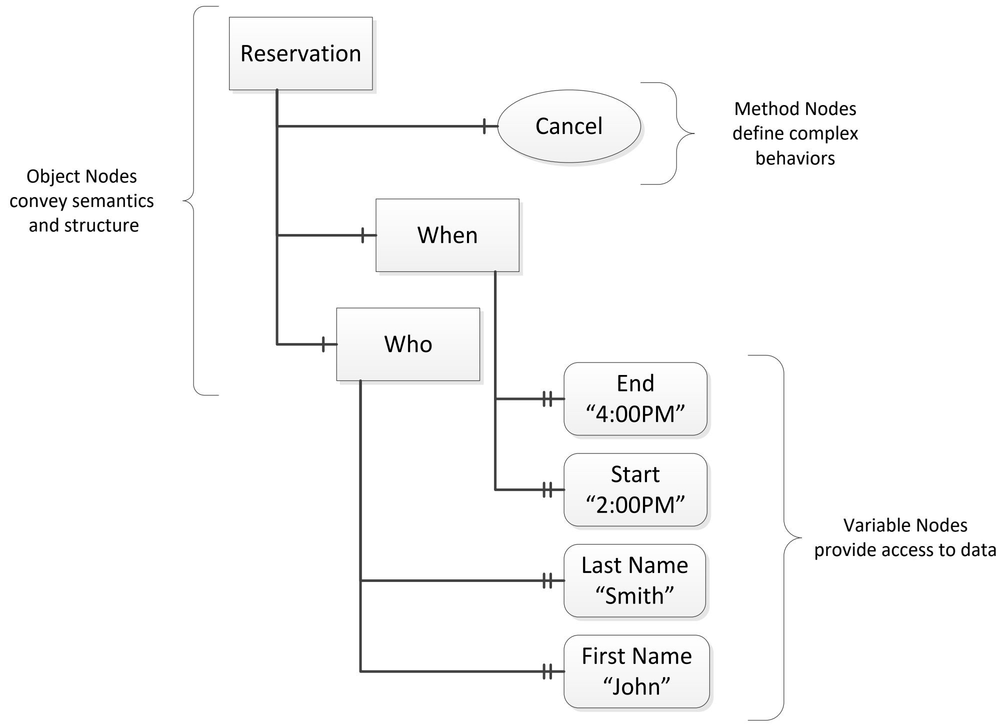


Figure 4 – A Basic Object in an OPC UA Address Space

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 identifies 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 10000-3, 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.

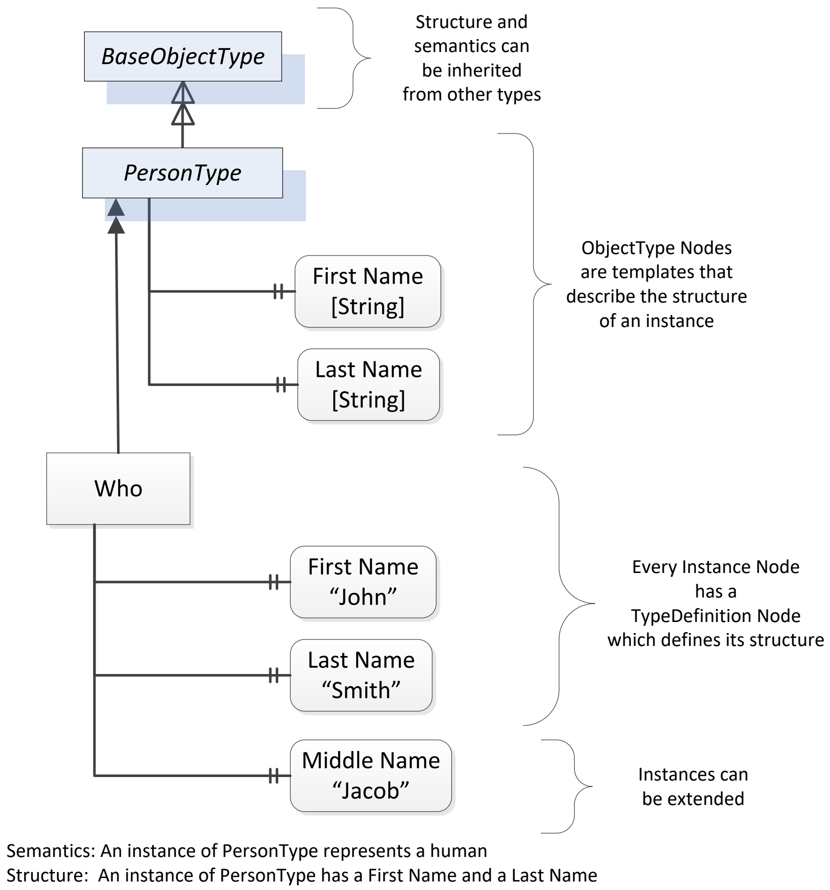


Figure 5 – The Relationship between Type Definitions and Instances

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*.

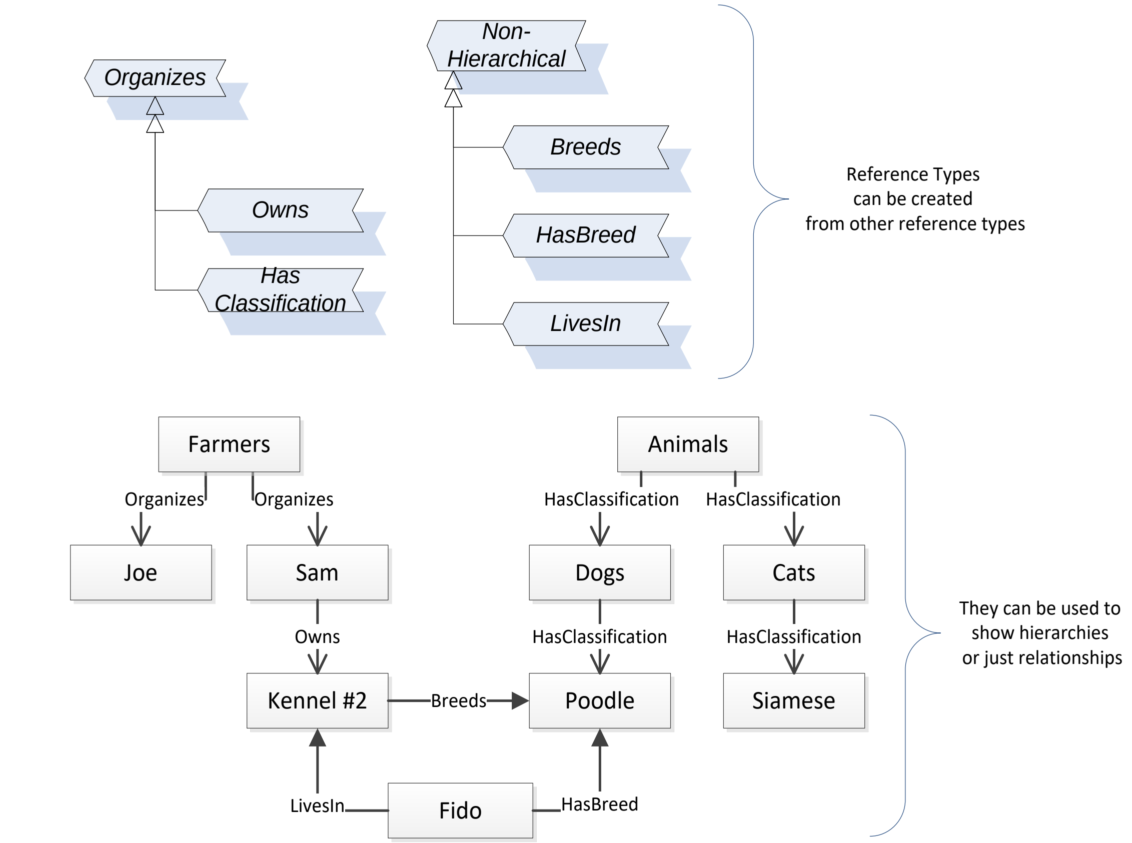


Figure 6 – Examples of References between Objects

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*.

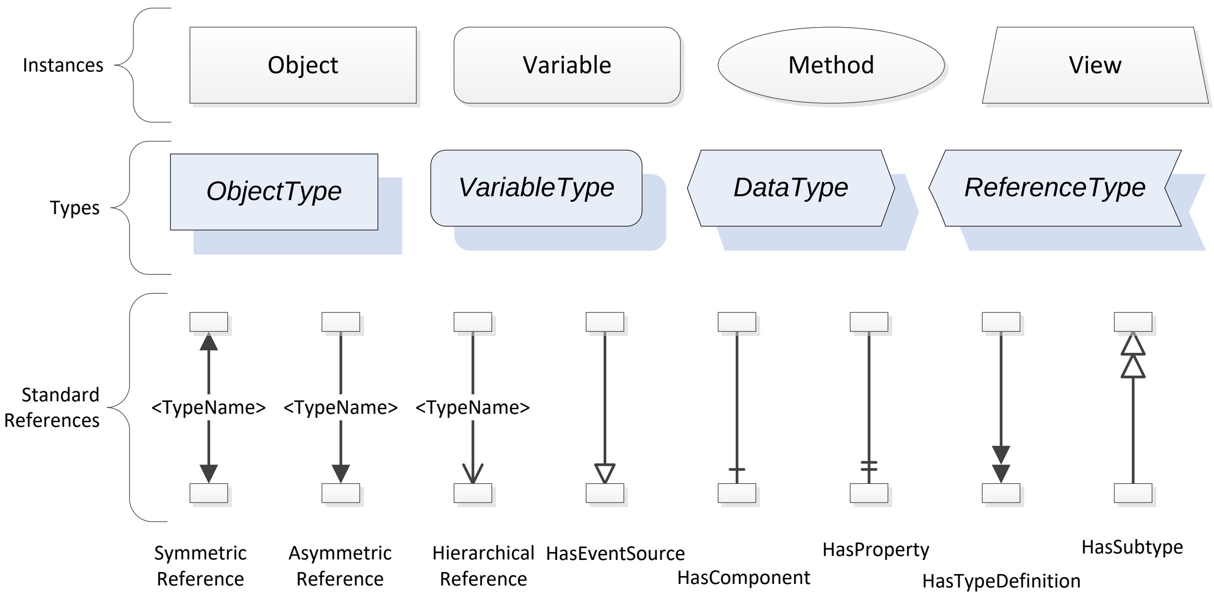


Figure 7 – The OPC UA Information Model Notation

A complete description of the different types of Nodes and References can be found in OPC 10000-3 and the base structure is described in OPC 10000-5.

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 10000-7).

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 be supervised 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 maintenance 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 maintenance is possible because the technican knows in adcance which parts need to be changed and can be prepared.

Figure 8 shows the communication structure with OPC UA.

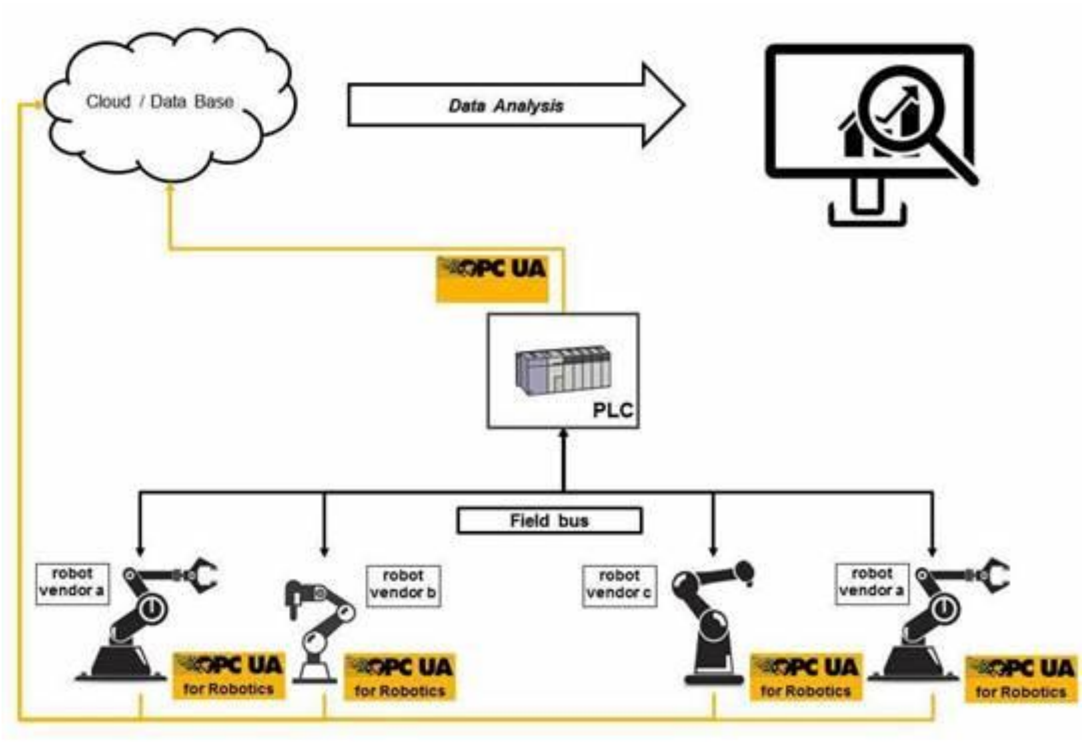


Figure 8 – Communication structure with OPC UA

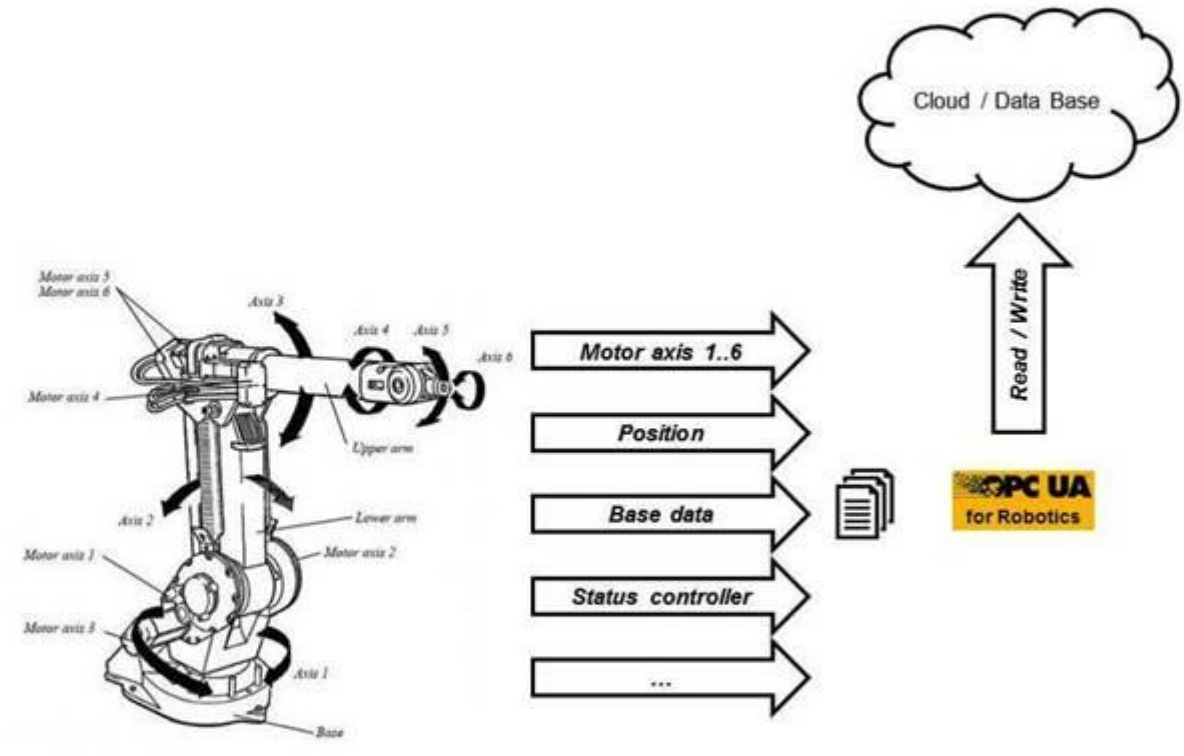


Figure 9 – OPC Robotics describes the semantic self-description

6 OPC Robotics Information Model overview

The *MotionDeviceSystemType* as a subtype of the *ComponentType* (OPC UA for Devices) is used as the root object representing the motion device system with all its subcomponents, see Figure 10.

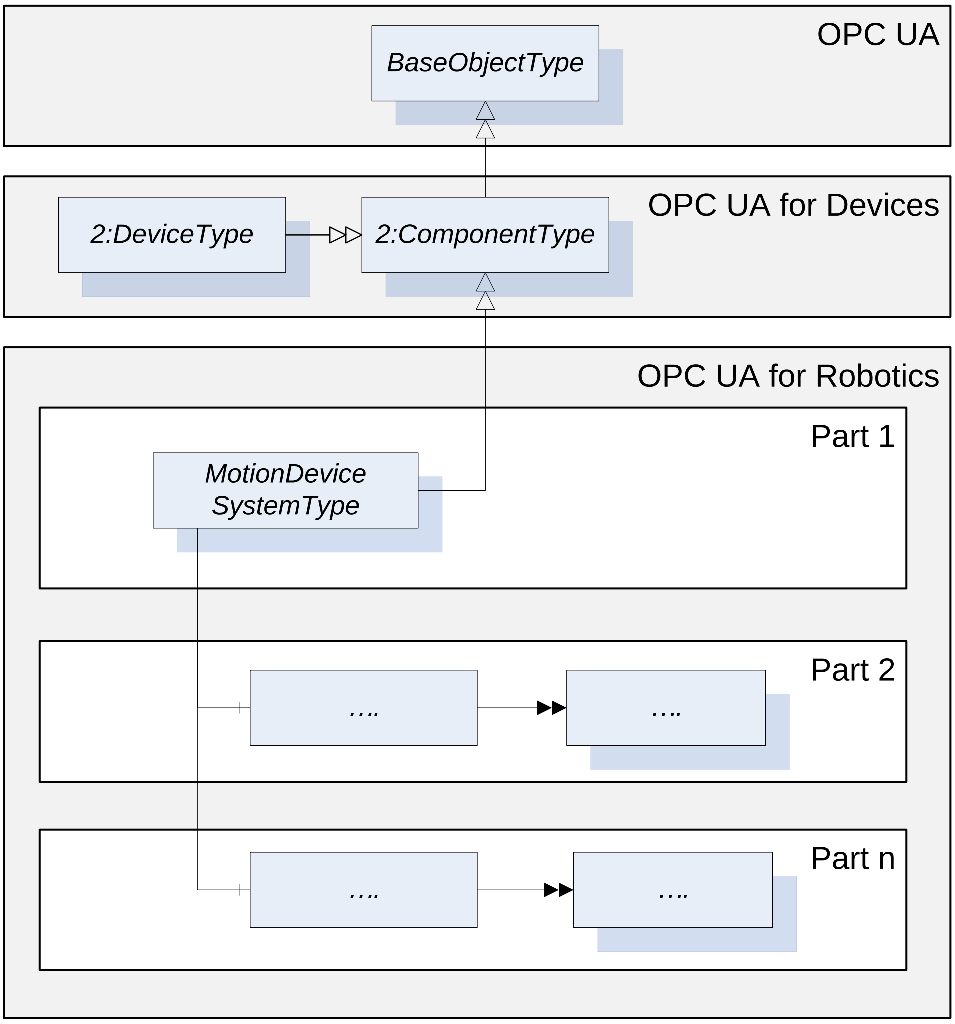


Figure 10 – OPC Robotics top level view

Figure 11 shows the main objects and the relations between them in an abstract view.

In Part 1 in general all variables and properties are read only unless stated otherwise in the description. A vendor can decide to provide variables or properties as writeable by client side as well.

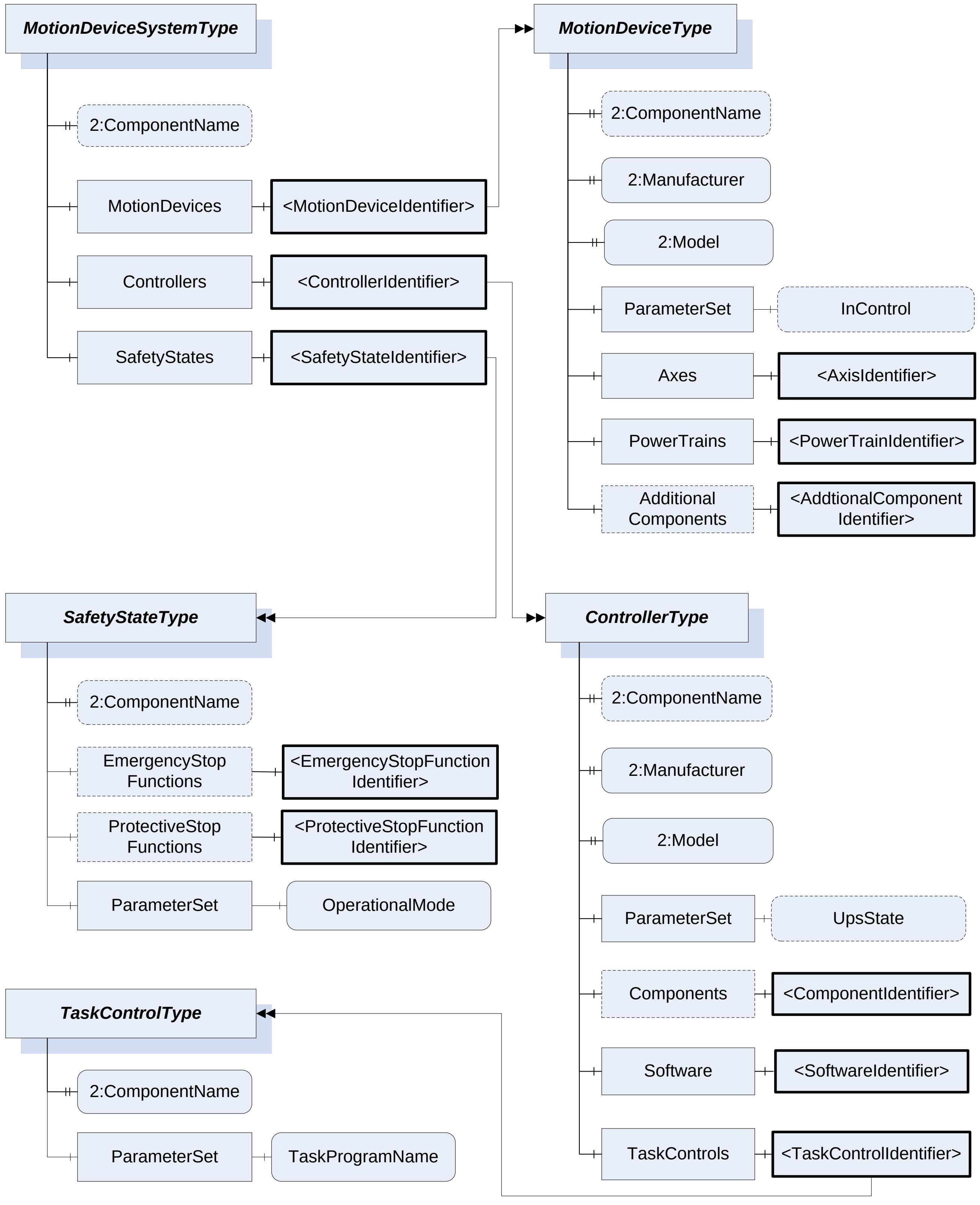


Figure 11 – OPC Robotics overview

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.

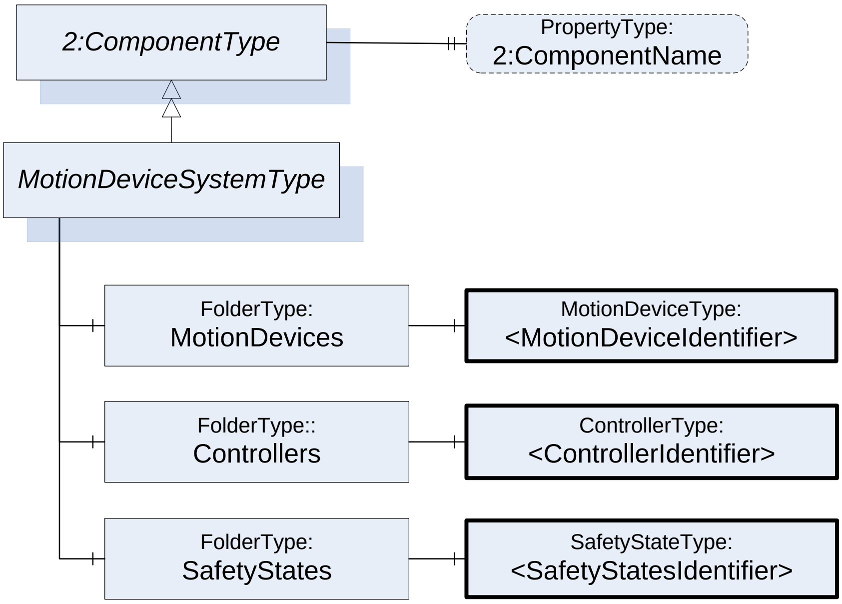


Figure 12 – Overview MotionDeviceSystemType

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 ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	MotionDevices		FolderType	Mandatory
HasComponent	Object	Controllers		FolderType	Mandatory
HasComponent	Object	SafetyStates		FolderType	Mandatory
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Optional

Table 12 – TypeDefinition of MotionDevices of MotionDeviceSystemType

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

Table 13 – TypeDefinition of Controllers of MotionDeviceSystemType

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

Table 14 – TypeDefinition of SafetyStates of MotionDeviceSystemType

Attribute	**Value**				
BrowseName	SafetyStates				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<SafetyStateIdentifier>		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 ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device. The *ComponentName* may be a default name given by the vendor. This property is defined by *ComponentType* defined in OPC UA DI.

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 and one power train. The *MotionDeviceType is* formally defined in Figure 14.

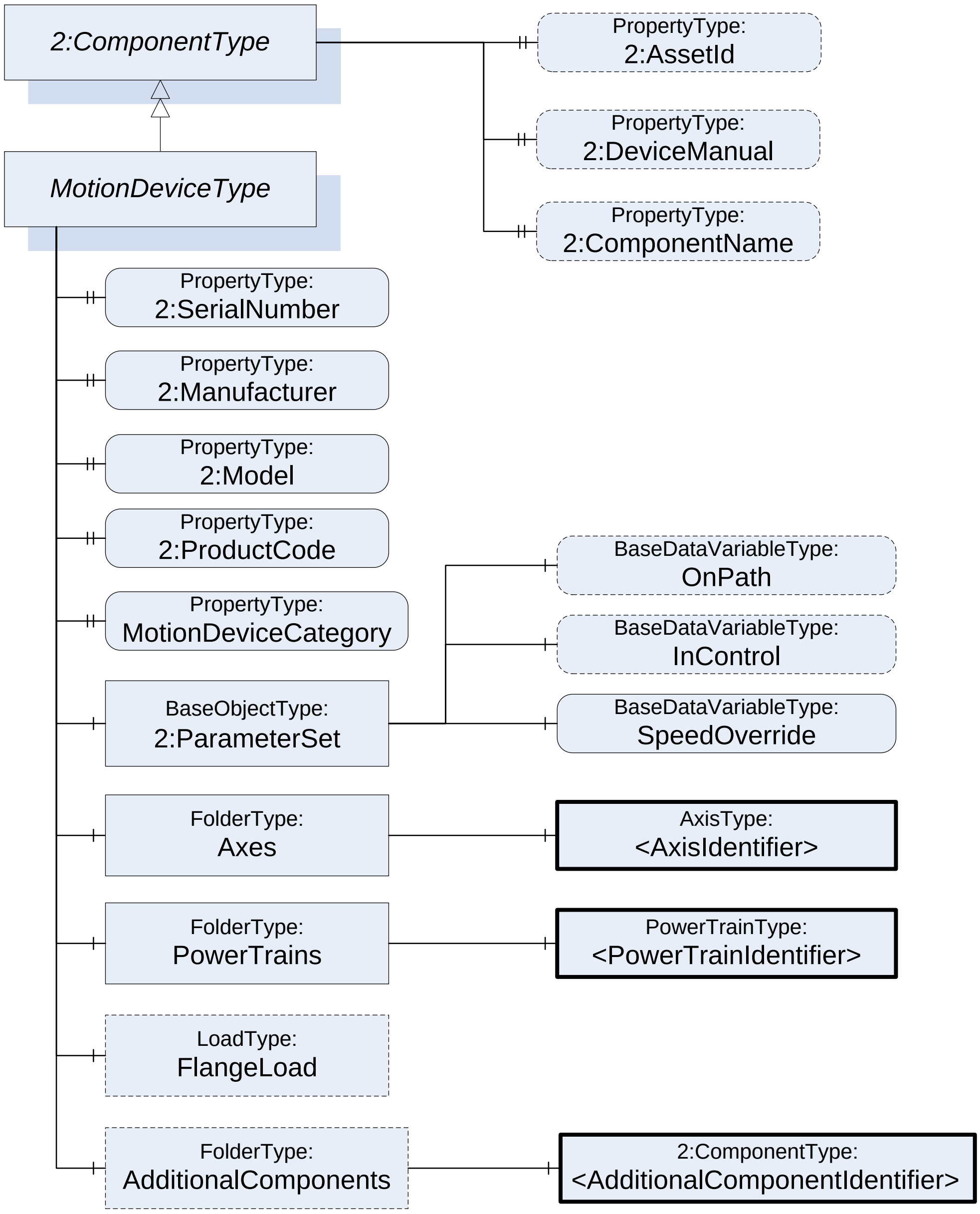


Figure 13 – Overview MotionDeviceType

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 ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	2:Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
HasProperty	Variable	MotionDeviceCategory	MotionDeviceCategoryEnumeration	PropertyType	Mandatory
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
HasComponent	Object	Axes		FolderType	Mandatory
HasComponent	Object	PowerTrains		FolderType	Mandatory
HasComponent	Object	FlangeLoad		LoadType	Optional
HasComponent	Object	AdditionalComponents		FolderType	Optional
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional
HasProperty	Variable	2:DeviceManual	String	PropertyType	Optional
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Optional

7.2.3 ObjectType description

7.2.3.1 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. This property is derived from *ComponentType* defined in OPC UA DI.

7.2.3.2 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the device. This property is derived from *ComponentType* defined in OPC UA DI.

7.2.3.3 Variable Model

The *Model* property provides the name of the product. This property is derived from *ComponentType* defined in OPC UA DI.

7.2.3.4 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.2.3.5 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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.

This property is defined by *ComponentType* defined in OPC UA DI.

7.2.3.6 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). This property is defined by *ComponentType* defined in OPC UA DI.

7.2.3.7 Variable ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device. The *ComponentName* may be a default name given by the vendor. This property is defined by *ComponentType* defined in OPC UA DI.

7.2.3.8 Variable MotionDeviceCategory

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

Table 16 – MotionDeviceCategoryEnumeration

MotionDeviceCategoryEnumeration		
EnumString	Value	Description
OTHER	0	Any motion-device which is not defined by the MotionDeviceCategoryEnumeration
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.

7.2.3.9 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%).

7.2.3.10 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	<AxisIdentifier>		AxisType	MandatoryPlaceholder

7.2.3.11 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	<PowerTrainIdentifier>		PowerTrainType	MandatoryPlaceholder

7.2.3.12 Object FlangeLoad

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

7.2.3.13 Object AdditionalComponents

AdditionalComponents is a container for one or more instances of subtypes of *ComponentType* defined in OPC UA DI. The listed components are installed at the motion device, e.g. an IO-board.

NOTE: Components like motors or gears of a motion device are placed inside the power train object and not inside this *AdditionalComponents* container.

Table 20 – TypeDefinition of AdditionalComponents of MotionDeviceType

Attribute	Value				
BrowseName	AdditionalComponents				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<AdditionalComponentIdentifier>		2:ComponentType	MandatoryPlaceholder

The AuxiliaryComponentType and DriveType are the only subtypes of *ComponentType* for use in this container which are described in this specification. The intention is to integrate inside this container devices which are defined in other companion specifications using DI.

7.3 AxisType ObjectType Definition

7.3.1 Overview

The *AxisType* describes an axis of a motion device. It is formally defined in Table 21.

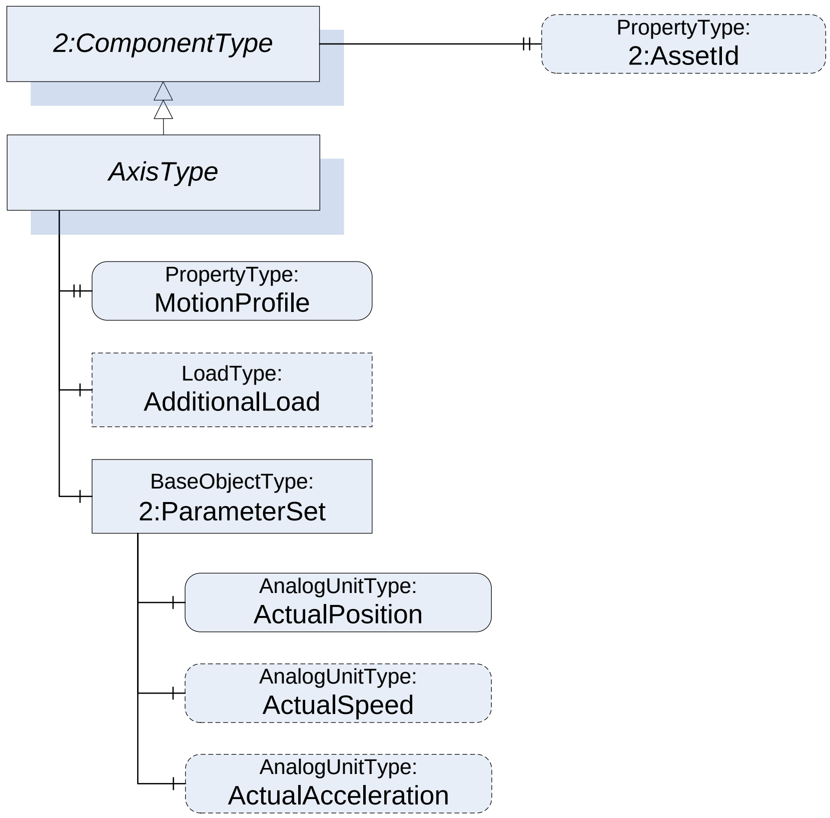


Figure 14 – Overview AxisType

7.3.2 ObjectType definition

Table 21 – AxisType Definition

Attribute	**Value**				
BrowseName	AxisType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	MotionProfile	AxisMotionProfileEnumeration	PropertyType	Mandatory
HasComponent	Object	AdditionalLoad		LoadType	Optional
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
Requires	Object	<PowerTrainIdentifier>		PowerTrainType	OptionalPlaceholder
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional

7.3.3 ObjectType description

7.3.3.1 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

The AssetID of the AxisType provides a manufacturer-specific axis identifier within the control system.

This property is defined by *ComponentType* defined in OPC UA DI.

7.3.3.2 Variable MotionProfile

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

Table 22 – 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 Objekt ParameterSet

Table 23 – ParameterSet of AxisType

Attribute	**Value**				
BrowseName	AxisType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	ActualPosition	Double	AnalogUnitType	Mandatory
HasComponent	Variable	ActualSpeed	Double	AnalogUnitType	Optional
HasComponent	Variable	ActualAcceleration	Double	AnalogUnitType	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 *AnalogUnitType* 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 *AnalogUnitType*
- Variable *ActualAcceleration*: The *ActualAcceleration* variable provides the axis acceleration. Applicable acceleration limits of the axis shall be provided by the *EURange* property of the *AnalogUnitType*.

7.3.3.5 Reference Requires

The *Requires* 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. This reference connects all power trains to an axis that must be actively driven when *only this axis* should move and all other axes should stand still.

Virtual axes that are not actively driven by a power train do not have this reference. The *InverseName* is *IsRequiredBy*.

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 24. A power train typically consists of one motor and gear to provide the required torque. 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.

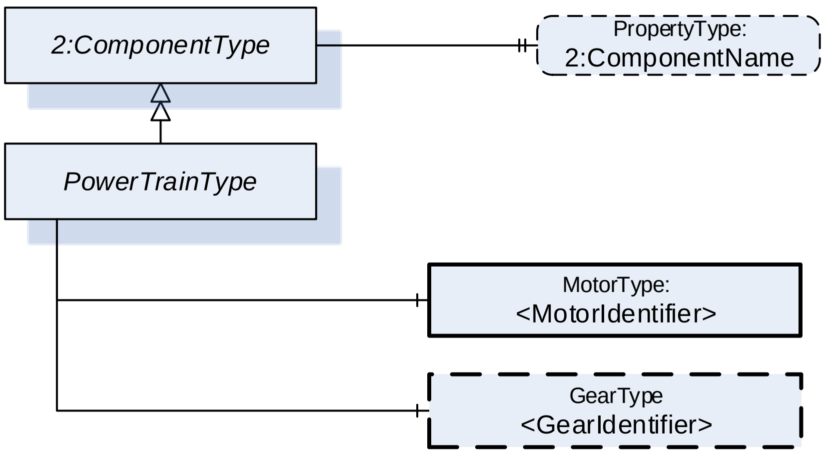


Figure 15 – Overview PowerTrainType

7.4.2 ObjectType definition

Table 24 – PowerTrainType Definition

Attribute	Value				
BrowseName	PowerTrainType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	<MotorIdentifier>		MotorType	MandatoryPlaceholder
HasComponent	Object	<GearIdentifier>		GearType	OptionalPlaceholder
Moves	Object	<AxisIdentifier>		AxisType	OptionalPlaceholder
HasSlave	Object	<PowerTrainIdentifier>		PowerTrainType	OptionalPlaceholder
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Optional

7.4.3 ObjectType description

7.4.3.1 Variable ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device. The *ComponentName* may be a default name given by the vendor.

The *ComponentName* of the PowerTrainType provides a manufacturer-specific power train identifier within the control system.

This property is defined by *ComponentType* defined in OPC UA DI.

7.4.3.2 Object <MotorIdenfifier>

<*MotorIdentifier*> indicates that a power train contains one or more motors represented by *MotorType* instances.

7.4.3.3 Object <GearIdentifier>

<*GearIdentifier*> indicates that a power train may contain one or more gears represented by *GearType* instances.

7.4.3.4 Reference Moves

Moves is a reference to provide the relationship of power trains to axes. For complex kinematics this does not need to be a one to one relationship, because a power train might influence the motion of more than one axis. This reference connects all axis to a power train that that move when *only this power train* moves and all other powertains stand still.

The *InverseName* is *IsMovedBy*.

7.4.3.5 Reference HasSlave

HasSlave is a reference to provide the master-slave relationship of power trains which provide torque for a common axis. The *InverseName* is *IsSlaveOf*.

7.5 MotorType ObjectType Definition

7.5.1 Overview

The *MotorType* describes a motor in a power train. It is formally defined in Table 25.

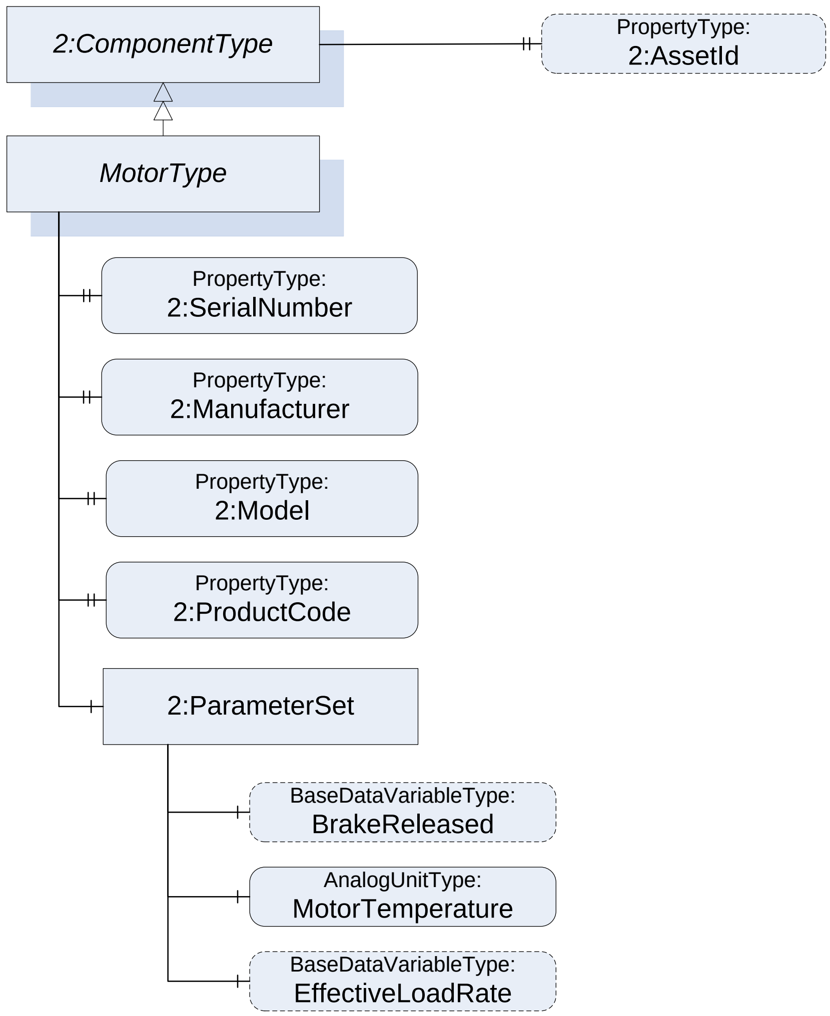


Figure 16 – Overview MotorType

7.5.2 ObjectType definition

Table 25 – MotorType Definition

Attribute	Value				
BrowseName	MotorType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	2:Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
IsConnectedTo	Object	<GearIdentifier>		GearType	OptionalPlaceholder
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
IsDrivenBy	Object	<DriveIdentifiier>		BaseObjectType	OptionalPlaceholer
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional

7.5.3 ObjectType description

7.5.3.1 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. This property is derived from *ComponentType* defined in OPC UA DI.

7.5.3.2 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the device. This property is derived from *ComponentType* defined in OPC UA DI.

7.5.3.3 Variable Model

The *Model* property provides the name of the product. This property is derived from *ComponentType* defined in OPC UA DI.

7.5.3.4 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.5.3.5 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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.

This property is defined by *ComponentType* defined in OPC UA DI.

7.5.3.6 Reference IsConnectedTo

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

7.5.3.7 Reference IsDrivenBy

IsDrivenBy is a reference to provide a relationship from a motor to a drive, which can be a multi-slot-drive or single slot drive*. The TypeDefinition* of the reference destination as *BaseObjectType* provides the possibility to point to a slot of a mulit-slot-drive or a motor-integrated-drive. If this reference points to a physical drive (and not a drive slot) it should point to an *DriveType*.

Annex B.1.9 shows different possibilities of usage.

7.5.3.8 Object ParameterSet

Table 26 – ParameterSet of MotorType

Attribute	Value				
BrowseName	ParameterSet				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Variable	BrakeReleased	Boolean	BaseDataVariableType	Optional
HasComponent	Variable	MotorTemperature	Double	AnalogUnitType	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.:

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

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 27.

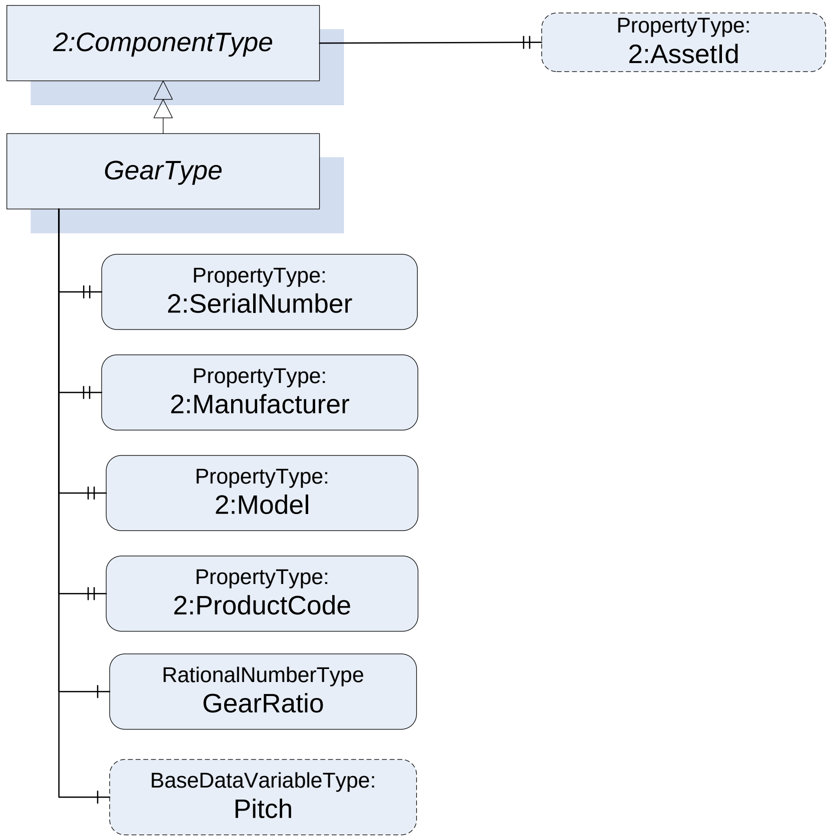


Figure 17 – Overview GearType

7.6.2 ObjectType definition

Table 27 – GearType Definition

Attribute	Value				
BrowseName	GearType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	2:Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
HasComponent	Variable	GearRatio	RationalNumber	RationalNumberType	Mandatory
HasComponent	Variable	Pitch	Double	BaseDataVariableType	Optional
IsConnectedTo	Object	<MotorIdentifier>		MotorType	OptionalPlaceholder
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	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 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. This property is derived from *ComponentType* defined in OPC UA DI.

7.6.3.2 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the device. This property is derived from *ComponentType* defined in OPC UA DI.

7.6.3.3 Variable Model

The *Model* property provides the name of the product. This property is derived from *ComponentType* defined in OPC UA DI.

7.6.3.4 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.6.3.5 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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. This property is defined by *ComponentType* defined in OPC UA DI.

7.6.3.6 Variable GearRatio

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

RationalNumberType and *RationalNumber* are defined in the OPC 10001-11 (SpatialTypes).

7.6.3.7 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:

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

7.6.3.8 Reference IsConnectedTo

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

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 28.

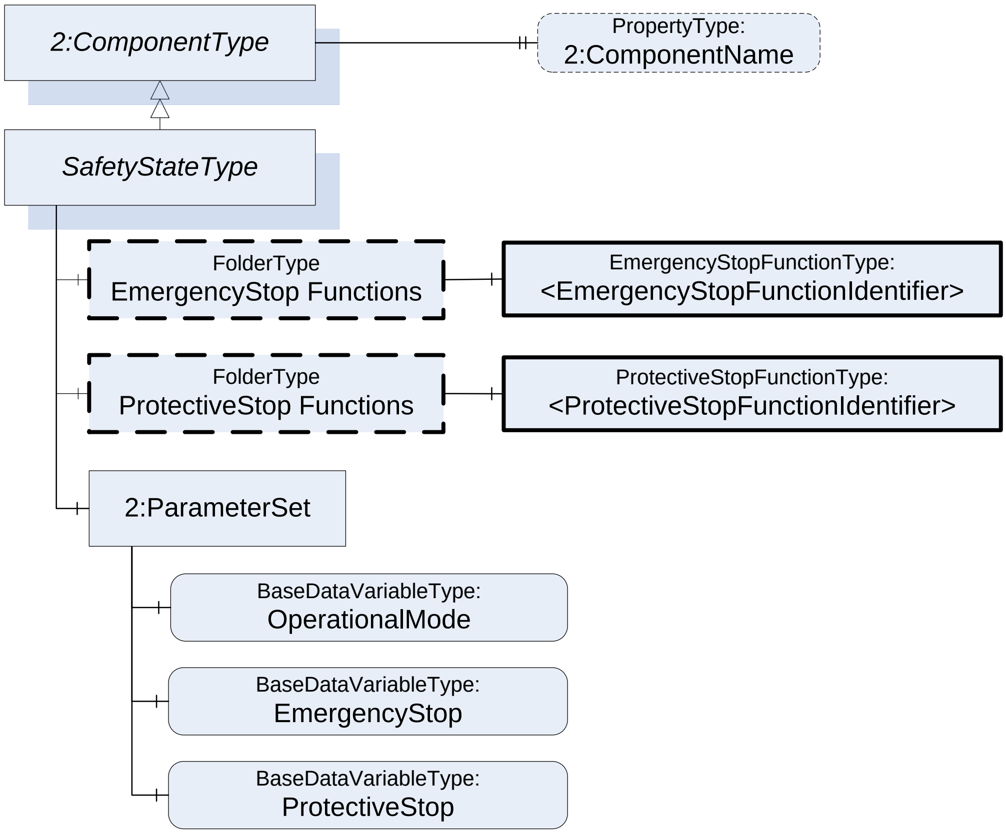


Figure 18 – Overview SafetyStateType

7.7.2 ObjectType definition

Table 28 – SafetyStateType Definition

Attribute	**Value**				
BrowseName	SafetyStateType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasComponent	Object	EmergencyStopFunctions		FolderType	Optional
HasComponent	Object	ProtectiveStopFunctions		FolderType	Optional
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Optional

7.7.3 ObjectType description

7.7.3.1 Variable ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device. The *ComponentName* may be a default name given by the vendor. This property is defined by *ComponentType* defined in OPC UA DI.

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 29 – TypeDefinition of EmergencyStopFunctions of SafetyStateType

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

Table 30 – 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.

- The *Name* of the EmergencyStopFunctionType provides a manufacturer-specific 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.

– *The Active* variable 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 31 – TypeDefinition of ProtectiveStopFunctions of SafetyStateType

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

Table 32 – 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.

– The *Name* of the ProtectiveStopFunctionType provides a manufacturer-specific protective stop function identifier within the safety system.

– The *Enabled* variable 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.

– The *Active* variable 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 33 – 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 34 – 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 Object ParameterSet

Table 35 – 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*: The *OperationalMode* variable 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*: The *EmergencyStop* variable 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*: The *ProtectiveStop* variable 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 36 – 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	"Manual 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 37.

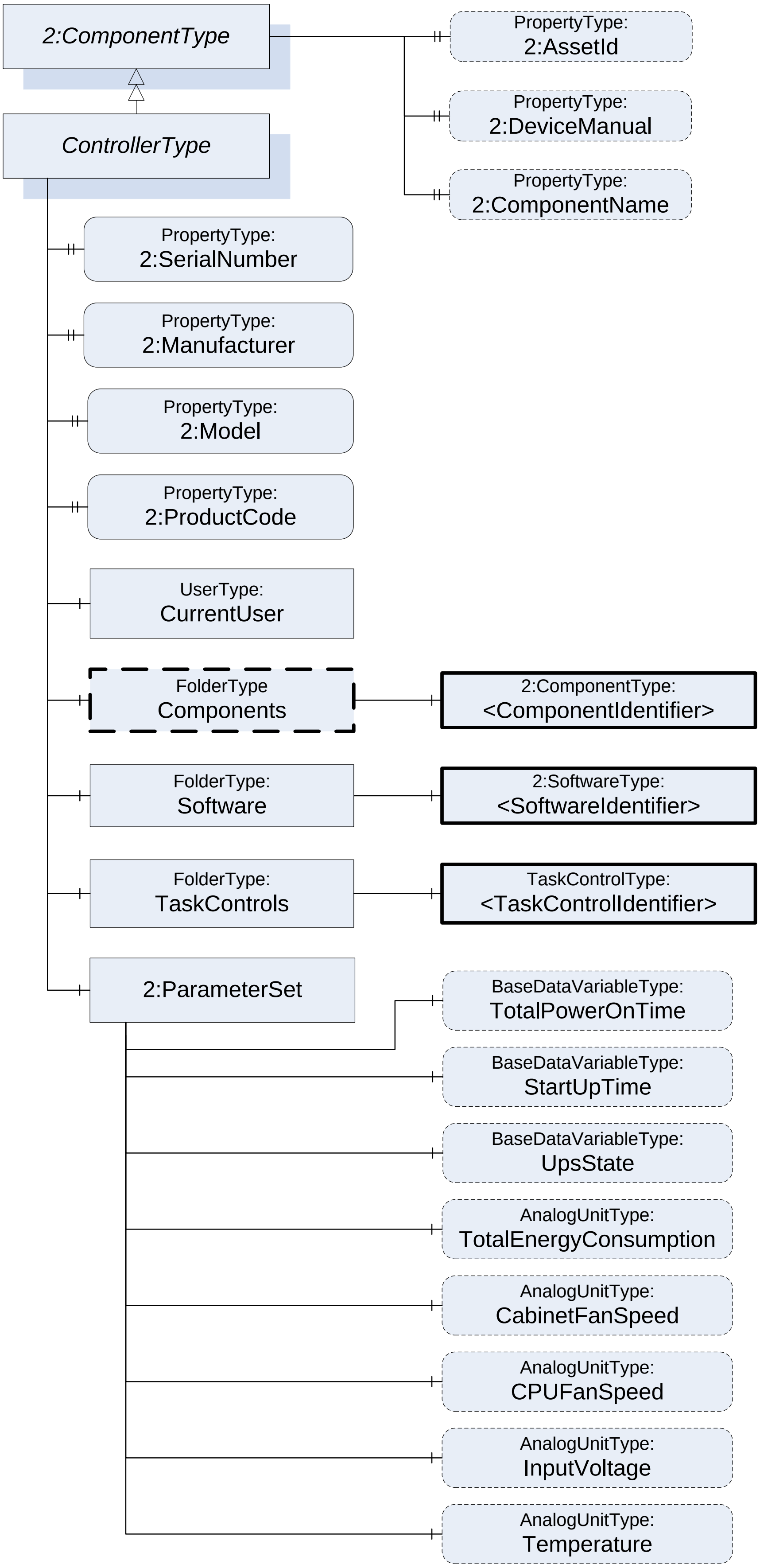


Figure 19 – Overview ControllerType

7.8.2 ObjectType definition

Table 37 – ControllerType Definition

Attribute	Value				
BrowseName	ControllerType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	2:Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
HasComponent	Object	CurrentUser		UserType	Mandatory
HasComponent	Object	Components		FolderType	Optional
HasComponent	Object	Software		FolderType	Mandatory
HasComponent	Object	TaskControls		FolderType	Mandatory
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
HasSafetyStates	Object	<SafetyStatesIdentifier>		SafetyStateType	OptionalPlaceholder
Controls	Object	<MotionDeviceIdentifier>		MotionDeviceType	OptionalPlaceholder
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional
HasProperty	Variable	2:DeviceManual	String	PropertyType	Optional
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Optional

7.8.3 ObjectType description

7.8.3.1 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. This property is derived from *ComponentType* defined in OPC UA DI.

7.8.3.2 Variable Manufacturer

The *Manufacturer* property provides the name of the company that manufactured the device. This property is derived from *ComponentType* defined in OPC UA DI.

7.8.3.3 Variable Model

The *Model* property provides the name of the product. This property is derived from *ComponentType* defined in OPC UA DI.

7.8.3.4 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.8.3.5 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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.

This property is defined by *ComponentType* defined in OPC UA DI.

7.8.3.6 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). This property is defined by *ComponentType* defined in OPC UA DI.

7.8.3.7 Variable ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device. The *ComponentName* may be a default name given by the vendor. This property is defined by *ComponentType* defined in OPC UA DI.

7.8.3.8 Object CurrentUser

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

7.8.3.9 Object Components

Components is a container for one or more instances of subtypes of *ComponentType* defined in OPC UA DI. The listed components are installed in the motion device system, e.g. a processing-unit, a power-supply, an IO-board or a drive, and have an electrical interface to the controller.

Table 38 – TypeDefinition of Components of ControllerType

Attribute	Value				
BrowseName	Components				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<ComponentIdentifier>		2:ComponentType	MandatoryPlaceholder

The *AuxiliaryComponentType* and *DriveType* are the only subtypes of *ComponentType* for use in this container which are described in this specification. The intention is to integrate inside this container devices which are defined in other companion specifications using DI.

7.8.3.10 Object Software

Software is a container for one or more instances of *SoftwareType* defined in OPC UA DI.

Each controller has at least one software installed.

Table 39 – TypeDefinition of Software of ControllerType

Attribute	Value				
BrowseName	Software				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<SoftwareIdentifier>		2:SoftwareType	MandatoryPlaceholder

7.8.3.11 Object TaskControls

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

Table 40 – TypeDefinition of TaskControls of ControllerTyp

Attribute	Value				
BrowseName	TaskControls				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<TaskControlIdentifier>		TaskControlType	MandatoryPlaceholder

7.8.3.12 Object ParameterSet

Table 41 – 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	AnalogUnitType	Optional
HasComponent	Variable	CabinetFanSpeed	Double	AnalogUnitType	Optional
HasComponent	Variable	CPUFanSpeed	Double	AnalogUnitType	Optional
HasComponent	Variable	InputVoltage	Double	AnalogUnitType	Optional
HasComponent	Variable	Temperature	Double	AnalogUnitType	Optional

Description of *ParameterSet* of *ControllerType*:

- Variable :The *TotalPowerOnTime* variable provides the total accumulated time the controller was powered on.
- Variable *StartUpTime*: The *StartUpTime* variable provides the date and time of the last start-up of the controller.
- Variable *UpsState*: The *UpsState* variable provides the vendor specific status of an integrated uninterruptible power supply or accumulator system.
- Variable *TotalEnergyConsumption*: The *TotalEnergyConsumption* variable provides total accumulated energy consumed by the motion devices related with this controller instance.
- Variable *CabinetFanSpeed*: The *CabinetFanSpeed* variable provides the speed of the cabinet fan.
- Variable *CPUFanSpeed*: The *CPUFanSpeed* variable provides the speed of the CPU fan.
- Variable *InputVoltage*: The *InputVoltage* variable provides the input voltage of the controller which can be a configured value. To distinguish between an AC or DC supply the optional property *Definition* of the base type *DataItemType* shall be used.
- Variable *Temperature*: The *Temperature* variable provides the controller temperature given by a temperature sensor inside of the controller.

7.8.3.13 Reference HasSafetyStates

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

7.8.3.14 Reference Controls

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

7.9 AuxiliaryComponentType ObjectType Definition

7.9.1 Overview

The *AuxiliaryComponentType* describes components mounted in a controller cabinet or a motion device e.g. an IO-board or a power supply.

It is formally defined in Table 42.

This type should not be used for instances of components which represent a motor, a gear or a drive For these components this specification describes specific types.

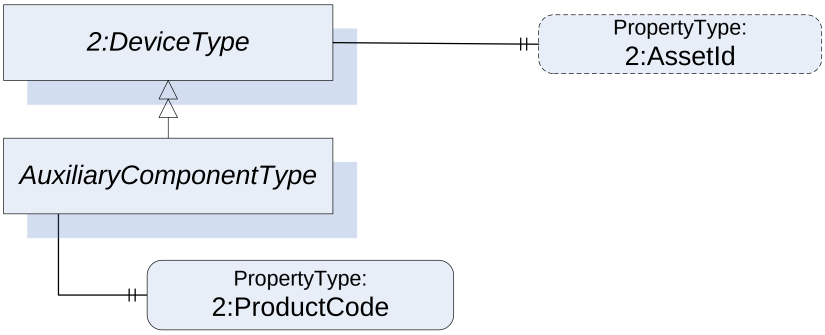


Figure 20 – Overview AuxiliaryComponentType

7.9.2 ObjectType definition

Table 42 – AuxiliaryComponentType Definition

Attribute	**Value**				
BrowseName	AuxiliaryComponentType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the DeviceType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional

7.9.3 ObjectType description

7.9.3.1 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.9.3.2 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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.

This property is defined by *ComponentType* defined in OPC UA DI.

7.10 DriveType

7.10.1 Overview

The *DriveType* describes drives (multi-slot or single-slot axis amplifier) mounted in a controller cabinet or a motion device. When used inside a motion device it should be part of a power train. It is formally defined in Table 42.

Annex B.1.9 shows different possibilities of usage.

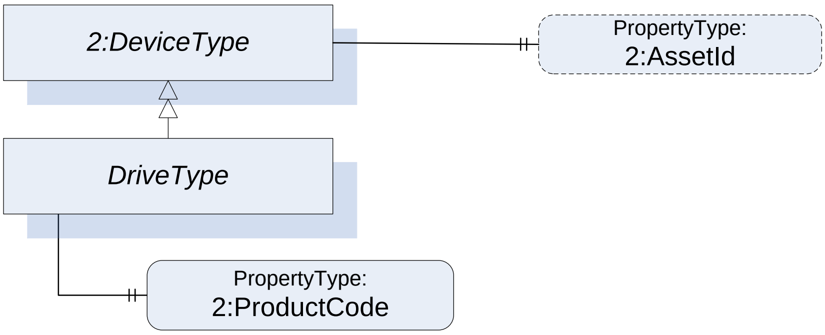


Figure 21 – Overview DriveType

7.10.2 ObjectType definition

Table 43 – DriveType Definition

Attribute	Value				
BrowseName	DriveType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the DeviceType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:ProductCode	String	PropertyType	Mandatory
The following instance declarations are not defined by this type, but by the supertype ComponentType and repeated here for better readability					
HasProperty	Variable	2:AssetId	String	PropertyType	Optional

7.10.3 ObjectType description

7.10.3.1 Variable ProductCode

The *ProductCode* property provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems. This property is derived from *ComponentType* defined in OPC UA DI.

7.10.3.2 Variable AssetId

The *AssetId* property is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the vendor, integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme.

This could be for example a reference to an electric scheme. For electric schemes typically EN 81346-2 is used.

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.

This property is defined by *ComponentType* defined in OPC UA DI.

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 44.

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.

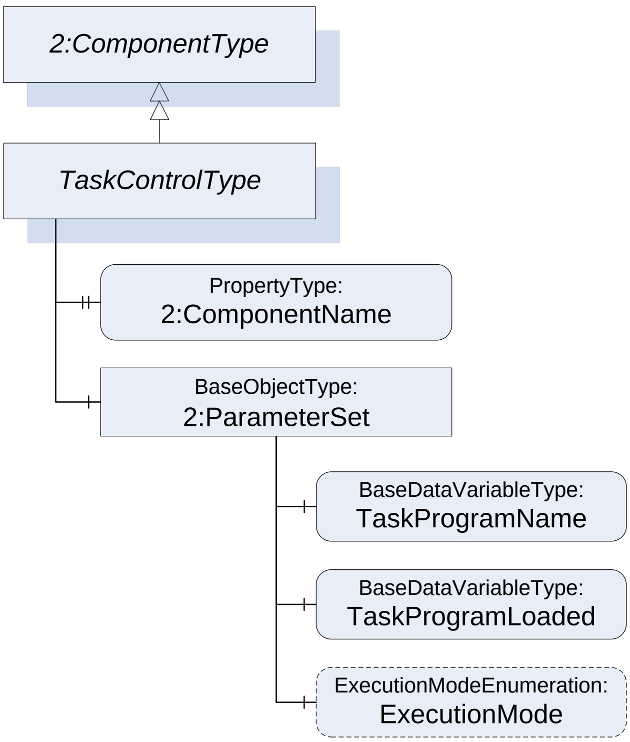


Figure 22 – Overview TaskControlType

7.11.2 ObjectType definition

Table 44 – TaskControlType Definition

Attribute	Value				
BrowseName	TaskControlType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ComponentType defined in OPC Unified Architecture for Devices (DI)					
HasProperty	Variable	2:ComponentName	LocalizedText	PropertyType	Mandatory
HasComponent	Object	2:ParameterSet		BaseObjectType	Mandatory
Controls	Object	<MotionDeviceIdentifier>		MotionDeviceType	OptionalPlaceholder

7.11.3 ObjectType description

7.11.3.1 Variable ComponentName

The *ComponentName* property provides a user writeable name provided by the vendor, integrator or user of the device.

The *ComponentName* of the *TaskControlType* provides a customer given identifier for the task control or a default name given by the vendor. This property is derived from *ComponentType* defined in OPC UA DI.

7.11.3.2 Object ParameterSet

Table 45 – 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* variable provides a customer given identifier for the task program.
- Variable *TaskProgramLoaded*: The *TaskProgramLoaded* variable is TRUE if a task program is loaded in the task control, FALSE otherwise.
- Variable *ExecutionMode*: The *ExecutionMode* variable tells how the task control executes the task program.

Table 46 – 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. The *InverseName* is *IsControlledBy.*

7.12 LoadType ObjectType Definition

7.12.1 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 47. 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.

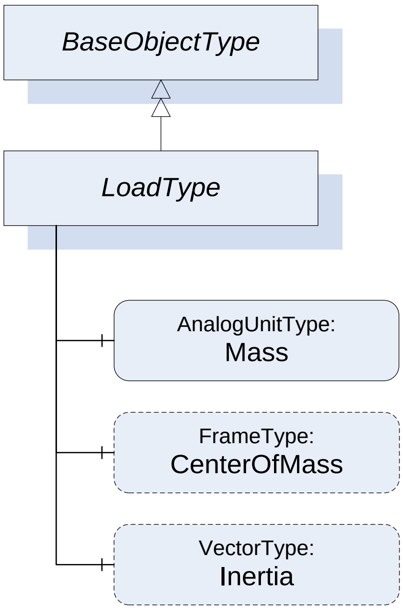


Figure 23 – Overview LoadType

7.12.2 ObjectType definition

Table 47 – 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	AnalogUnitType	Mandatory
HasComponent	Variable	CenterOfMass	3DFrame	3DFrameType	Optional
HasComponent	Variable	Inertia	3DVector	3DVectorType	Optional

7.12.3 ObjectType description

7.12.3.1 Variable Mass

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

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

7.12.3.2 Variable CenterOfMass

The variable *CenterOfMass* provides the position and orientation of the center of the mass related to the mounting point using a *3DFrameType*. 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 these values.

3DFrameType and *3DFrame* are defined in OPC 10001-11 (SpatialTypes).

If the instance of the *LoadType* describes the flange load of a motion device the mounting point coordinate system is the flange coordinate system. 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.

7.12.3.3 Variable Inertia

The variable *Inertia* uses the *3DVektorType* 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 48 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.

3DVectorType and *3DVector* are defined in OPC 10001-11 (SpatialTypes).

Table 48 – 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

7.13 UserType ObjectType Definition

7.13.1 Overview

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

It is formally defined in Table 49.

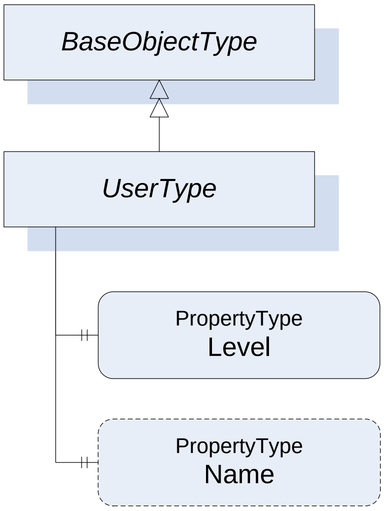


Figure 24 – Overview UserType

7.13.2 ObjectType definition

Table 49 – 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	PropertyType	Mandatory
HasProperty	Variable	Name	String	PropertyType	Optional

7.13.3 ObjectType description

7.13.3.1 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.

7.13.3.2 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 25 describes informally the hierarchy of these Reference Types. OPC UA Reference Types are defined in OPC 10000-3.

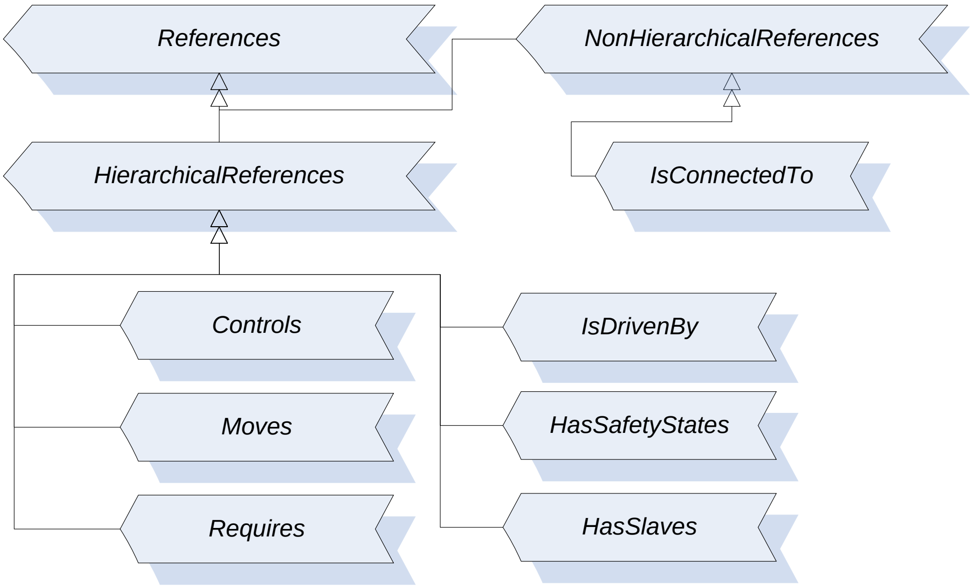


Figure 25 – Reference Type Hierarchy

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.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 50 – Controls Reference Definition

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

8.3 Moves (IsMovedBy) Reference Type

The OPC UA *ReferenceType Moves* is used to describe the coupling between a power train and the axes from the power train point of view. A power train has a *Moves* reference to all axis that are moving when only this powertrain moves.

For examples see Annex B.1.8.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 51 – Controls Reference Definition

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

8.4 Requires (IsRequiredBy) Reference Type

The OPC UA *ReferenceType Requires* is used to describe the coupling between a power train and axes from the axis point of view. An axis has a *Requires* reference to all powertrains that need to move such that only this single axis moves.

For examples see Annex B.1.8.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 52 – Controls Reference Definition

Attributes	Value				
BrowseName	Requires				
InverseName	IsRequiredBy				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferences defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.5 IsDrivenBy (Drives) Reference Type

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

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 53 – Drives Reference Definition

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

8.6 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.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 54 – IsConnectedTo Reference Definition

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

8.7 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.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 55 – HasSafetyStates Reference Definition

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

8.8 HasSlave (IsSlaveOf) Reference Type

The OPC UA *ReferenceType HasSlave* is a reference to provide the master-slave relationship of power trains which provide torque for a common axis. The *InverseName* is *IsSlaveOf*.

The *SourceNode* of this type shall be an *ObjectType* or *Object* and the *TargetNode* shall be an *Object*.

Table 56 – HasSlave Reference Definition

Attributes	**Value**				
BrowseName	HasSlave				
InverseName	IsSlaveOf				
Symmetric	False				
IsAbstract	False				
Subtype of the HierarchicalReferences defined in OPC Unified Architecture Part 5					
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

9 Profiles and Namespaces

9.1 Namespace Metadata

http://opcfoundation.org/UA/Robotics/ 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 Part 5 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 Part 5.

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 57 – NamespaceMetadata Object for this Specification

Attribute	**Value**		
BrowseName	http://opcfoundation.org/UA/Robotics/		
References	**BrowseName**	**DataType**	**Value**
HasProperty	NamespaceUri	String	http://opcfoundation.org/UA/Robotics/
HasProperty	NamespaceVersion	String	1.00
HasProperty	NamespacePublicationDate	DateTime	2019-02-04
HasProperty	IsNamespaceSubset	Boolean	Vendor-specific
HasProperty	StaticNodeIdTypes	IdType[]	Numeric
HasProperty	StaticNumericNodeIdRange	NumericRange[]	
HasProperty	StaticStringNodeIdPattern	String	

9.2 Conformance Units and Profiles

This chapter defines the corresponding *Profiles* and *Conformance Units* for the OPC UA Information Model for Robotics. *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.*

9.3 Server Profiles

The following tables specify the Profiles available for Robotic-Systems that implement the OPC UA for Robotics Information Model companion specification.

9.3.1 Robotics Base Profile

This Profile supports the information for Robotics. This Profile is intended to be used of OPC UA servers with limited resources. It is built upon the "Embedded 2017 UA Server Profile" Profile. The content of the Profile is defined in Table 72.

Table 58 – Robotics Base Profile

Conformance Unit	Description	Optional/ Mandatory
Robotics Part 1 ObjectTypes Mandatory	Supports all mandatory parts of the ObjectTypes that are defined in VDMA OPC Robotics Part 1.	M
Profiles		
Embedded 2017 UA Server Profile http://opcfoundation.org/UA-Profile/Server/EmbeddedUA2017		
BaseDevice_Server_Facet (defined in OPC 10000-100)		
"Data Access Server Facet" Profile http://opcfoundation.org/UA-Profile/Server/DataAccess		
"ComplexType 2017 Server Facet" Profile http://opcfoundation.org/UA-Profile/Server/ComplexTypes2017		

9.4 Client Facets

This specification does not define any Client Facets.

9.5 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 59 provides a list of mandatory and optional namespaces used in an Robotics OPC UA *Server*.

Table 59 – Namespaces used in a Robotics 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 Robotics 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 10000-100. The namespace index is server specific.	Mandatory
http://opcfoundation.org/UA/Robotics/	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 60 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 60 – Namespaces used in this specification

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

Annex A
(normative)

Robotics Namespace and mappings

A.1 Namespace and identifiers for Robotics 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 *<property> InstanceDeclaration* within the *<type>* declaration is: *AutoIdDeviceType_DeviceLocation*.

The *NamespaceUri* for all *NodeIds* defined here is http://opcfoundation.org/UA/Robotics/

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

– http://www.opcfoundation.org/UA/schemas/Robotics/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/Robotics/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/Robotics/1.0/Opc.Ua.Robotics.NodeSet2.xml

A.2 Profile URIs for Robotics Information Model

Table A.1 defines the Profile URIs for the Robotics Information Model companion specification.

Table A.1 – Profile URIs

Profile	Profile URI
First facet	http://opcfoundation.org/UA-Profile/External/Robotics/RoboticsBaseProfile
…	

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.

In addition, this chapter contains examples of how to use the references contained in this specification.

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 B.1, Figure B.2, Figure B.3, Figure B.4, Figure B.5, Figure B.6 and Figure B.7 normally have only one control unit.

Figure B.8 shows an example with four motion devices which can be controlled by one control unit.

The motion device system illustrated in Figure B.9 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 B.8 are typically controlled by one controller unit. Each motion device *IsControlledBy* the same controller.

The system illustrated in Figure B.9 may have two control units. For example one controller *Controls* the both articulated robots and the mobile platform *IsControlledBy the other controller*.

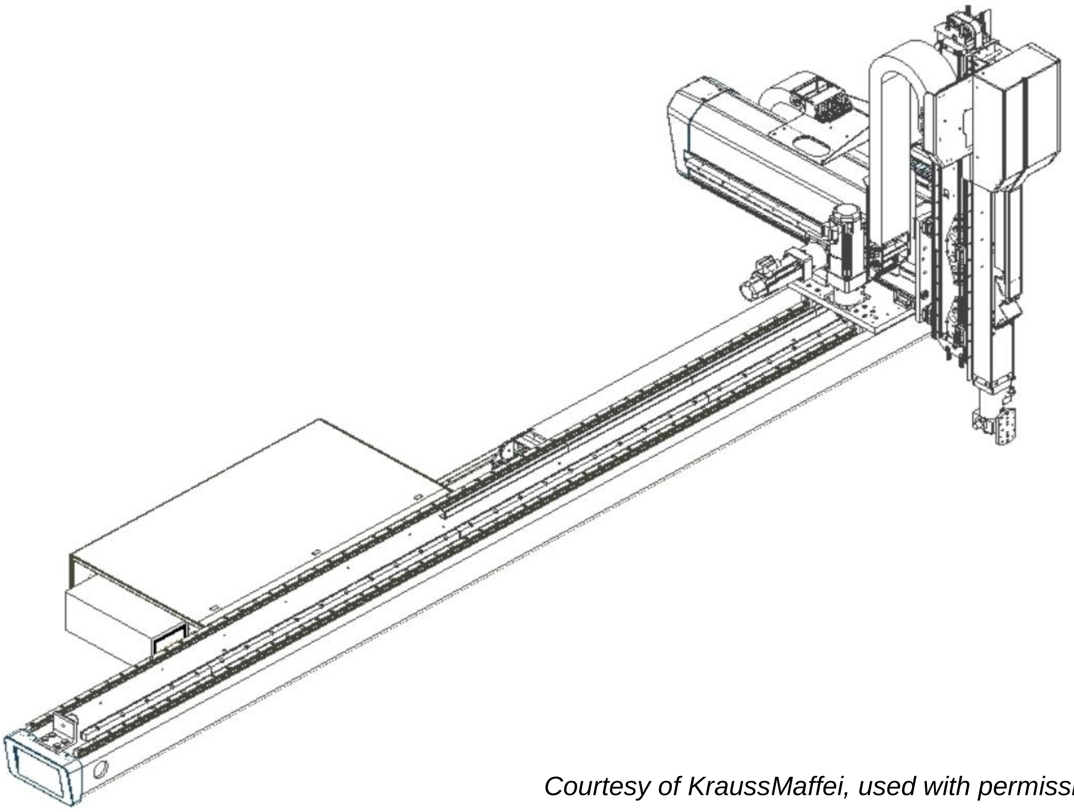
B.1.3 Examples for motion devices

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 MotionDeviceCategory* of the *ParameterSet* of the *MotionDeviceType* by the *MotionDeviceCategoryEnumeration,* which is based on definitions of ISO 8373:2012.

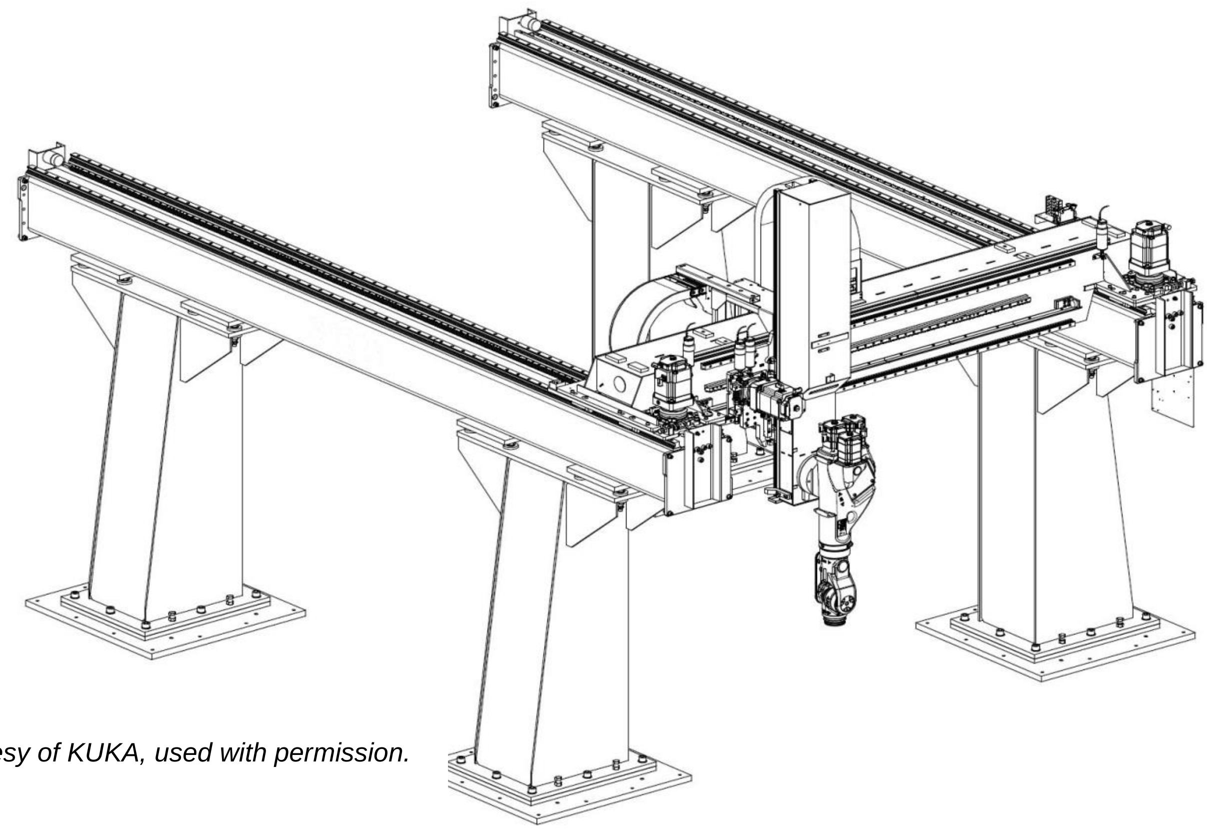
The Figures Figure B.1 and Figure B.2 show examples of cartesian manipulators.

Figure B.2 shows a portal manipulator, a variant of a cartesian manipulator. Axis 1 in this example is driven with master-slave and a robot-hand is mounted at the flange of the cartesian manipulator.



Courtesy of KraussMaffei, used with permission.

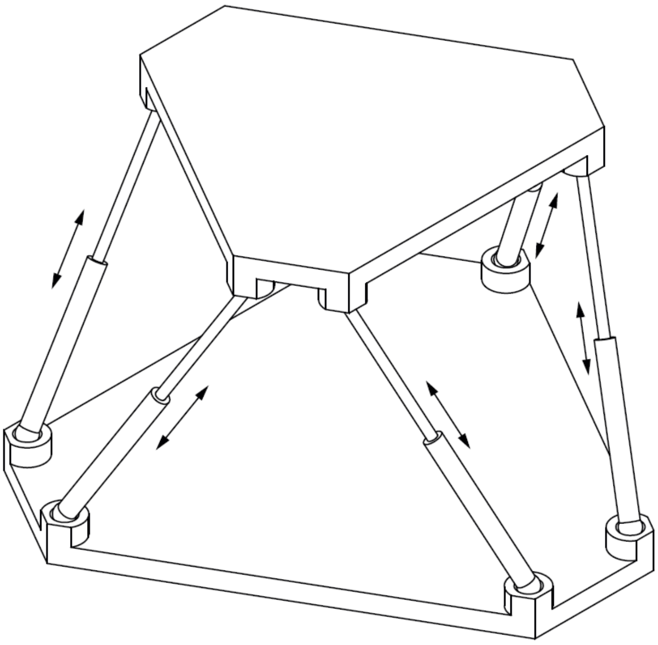
Figure B.1 – Cartesian manipulator



Courtesy of KUKA, used with permission.

Figure B.2 – Portal manipulator

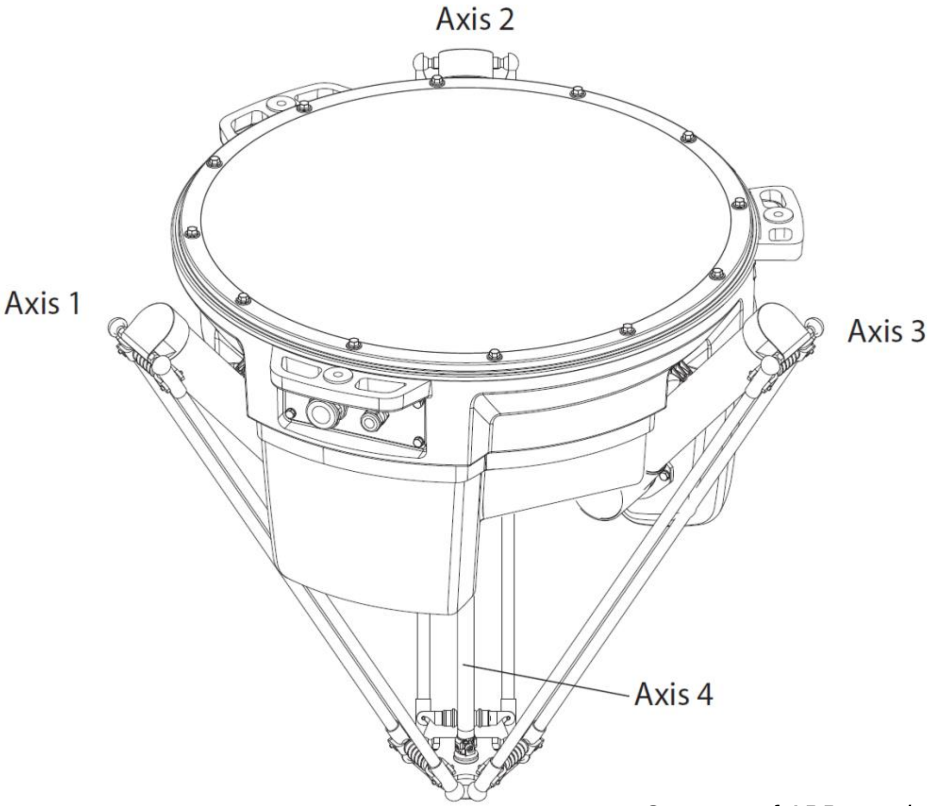
Figure B.3 shows an example of a parallel manipulator. So called delta robots, as shown in Figure B.4, are also parallel manipulators.



Courtesy of Beckhoff, used with permission.

Figure B.3 – Stewart platform or Hexapod

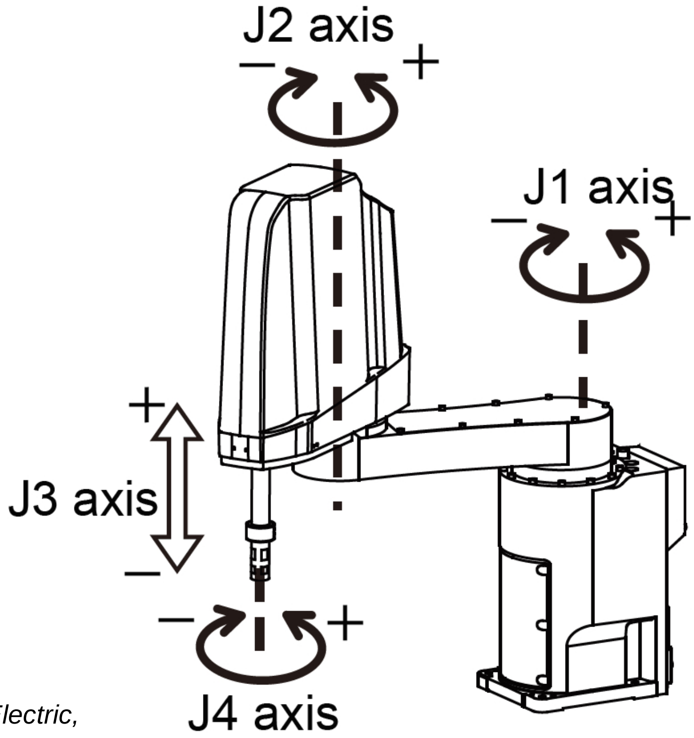
Figure B.4 shows an abstract example of a delta robot.



Courtesy of ABB, used with permission.

Figure B.4 – Delta robot

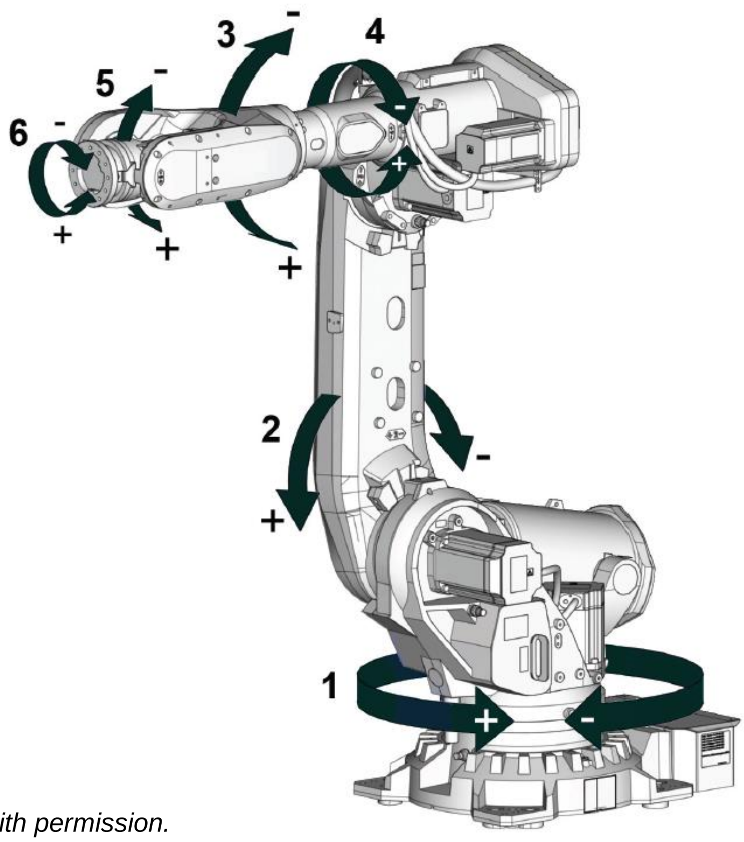
Figure B.5 shows an abstract example of a scara robot.



Courtesy of Mitsubishi Electric, used with permission.

Figure B.5 – Scara robot

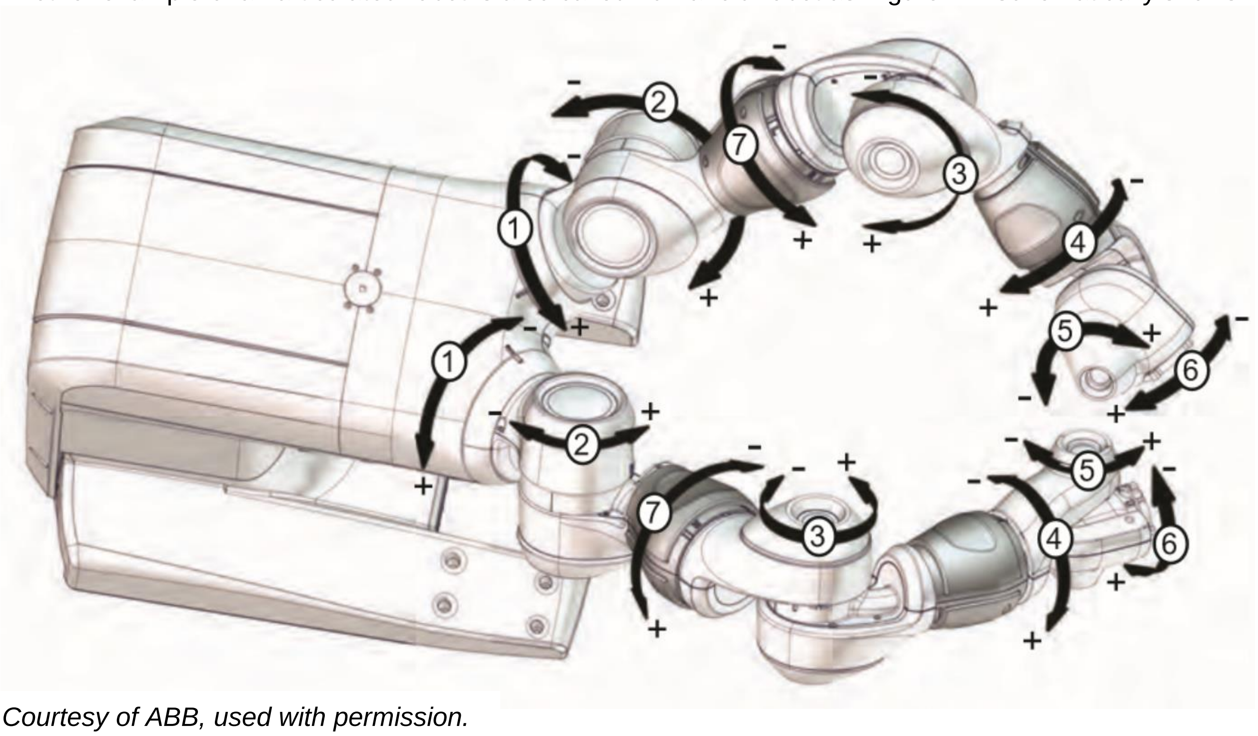
A typical example of an articulated robot is shown in Figure B.6.



Courtesy of ABB, used with permission.

Figure B.6 – Articulated robot

Another example of an articulated robot is a so called humanoid robot as Figure B.7 schematically shows.

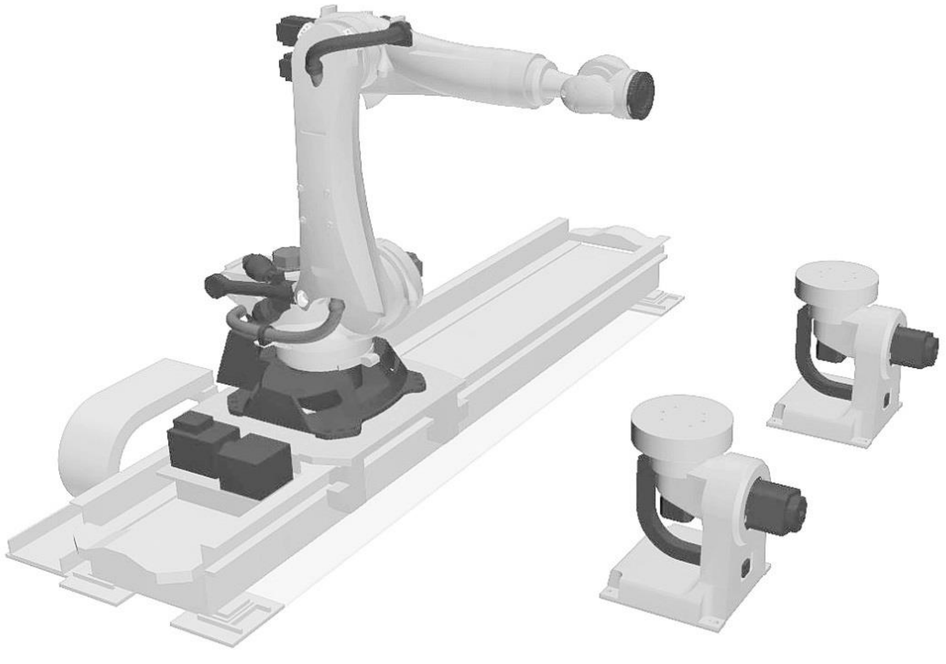


Courtesy of ABB, used with permission.

Figure B.7 – Schematic of a humanoid robot

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

Figure B.8 shows four motion devices integrated in one motion device system: an articulated robot on a linear unit with two turntables.



Courtesy of KUKA, used with permission.

Figure B.8 – Motion device system 1

Figure B.9 shows three motion devices in one motion device system: two articulated robots on a mobile platform.

Courtesy of KUKA, used with permission.

Figure B.9 – 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 B.10 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.

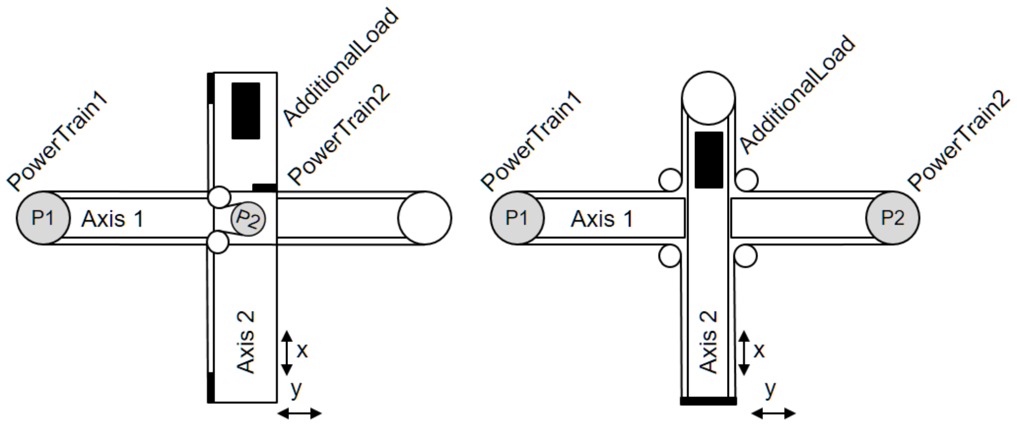


Figure B.10 – Axis and power train coupling

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

Figure B.1 and Figure B.2 show different versions of Cartesian robots. Figure B.1 shows a three axis robot which has one dedicated power train for each axis: A power train *Moves* exactly one axis and so an axis only *Requires* one dedicated power train. One motor of a power train *IsDrivenBy* a drive and *IsConnectedTo* a gear.

Figure B.2 shows a three axis robot with a master-slave driven axis 1. The first and second power train *Moves* axis 1. The first power train *HasSlave* the second power train. Axis 1 *Requires* the first and the second power train. For axis 2 and 3 one power train *Moves* exactly one axis and so an axis only *Requires* one dedicated power train.

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

B.1.8.1 Example articulated six-axis industrial robot

The typical six-axis industrial robot shown in Figure B.6 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. The complete set of references is depiced in Figure B.11.

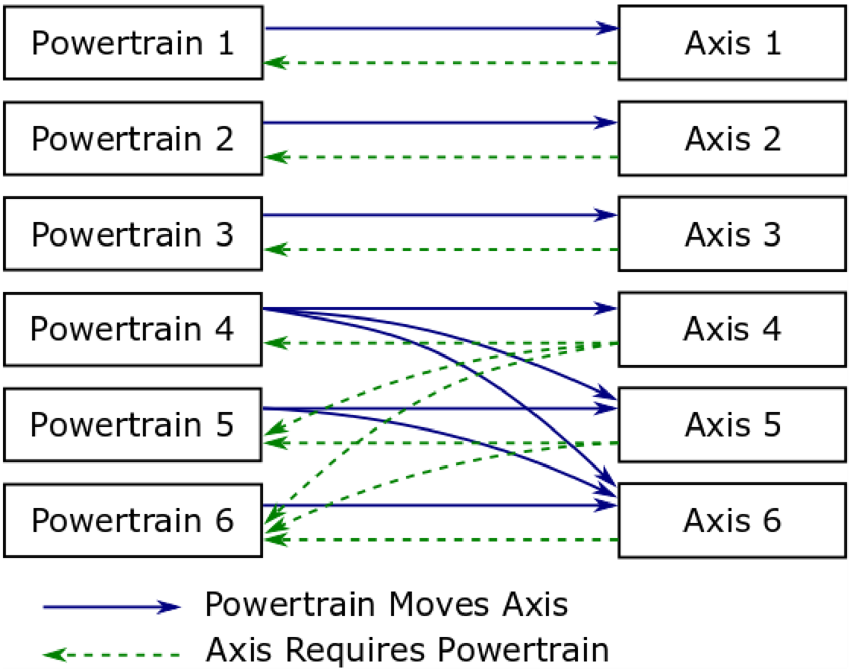


Figure B.11 – Coupling references for a typical six-axis industrial robot

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

i. Power train 1 *Moves* axis 1
ii. Power train 2 *Moves* axis 2
iii. Power train 3 *Moves* axis 3
iv. Power train 4 *Moves* axis 4, axis 5 and axis 6
v. Power train 5 *Moves* axis 5 and axis 6
vi. 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. It is the inverse of the *Moves* reference.

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 compensation movements of power train 5 and power train 6 are necessary 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. It is the inverse of the *Requires* reference.

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.

B.1.8.2 Example articulated six-axis industrial robot with 3 master-slave axes

A high-payload six-axis industrial robot shown in Figure B.6 can have nine power trains for the movement of the six axes. In this example the axes 1 to 3 are each driven by two power trains with master-slave configuration.

Figure B.12 shows the use of the *HasSlave* rerference in addition to the power train to axis references.

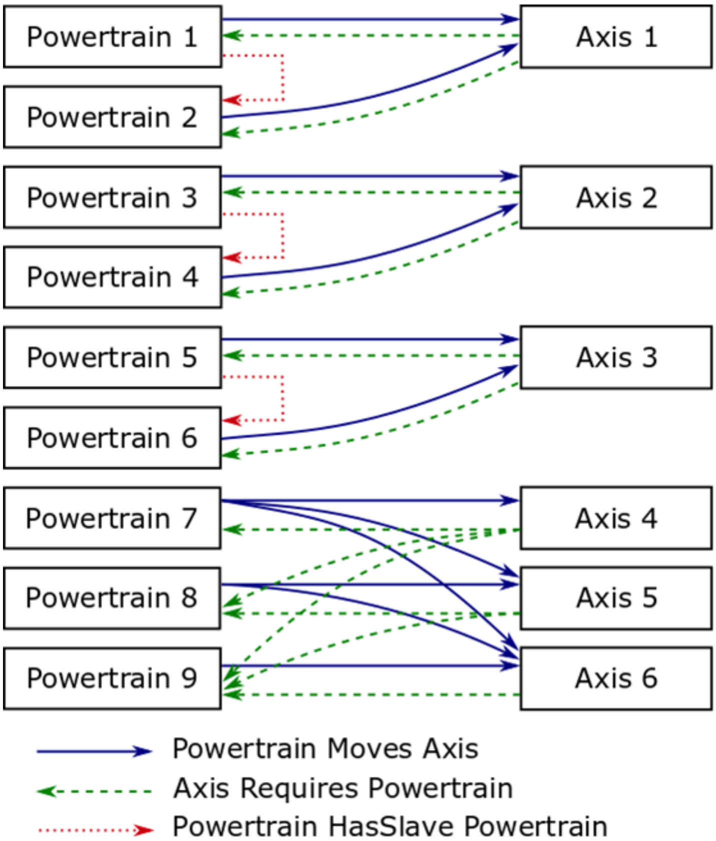


Figure B.12 – Coupling references for a six-axis industrial robot with master-slave axes

A power train HasSlave a power train means that one power train is the master of a master-slave-configuration and he references HasSlave to power train which is slave coupled.

HasSlave References:

i. Power train 1 *HasSlave* power train 2
ii. Power train 3 *HasSlave* power train 4
iii. Power train 5 *HasSlave* power train 6

For this master-slave configuration the *Moves* and Requires references :

i. Power train 1 *Moves* axis 1
ii. Power train 2 *Moves* axis 1
iii. Power train 3 *Moves* axis 2
iv. Power train 4 *Moves* axis 2
v. Power train 5 *Moves* axis 3
vi. Power train 6 *Moves* axis 3
vii. Power train 7 *Moves* axis 4, axis 5 and axis 6
viii. Power train 8 *Moves* axis 5 and axis 6
ix. Power train 9 *Moves* axis 6

i. Axis 1 *Requires* power train 1 and power train 2
ii. Axis 2 *Requires* power train 3 and power train 4
iii. Axis 3 *Requires* power train 5 and power train 6
iv. Axis 4 *Requires* power train 7, power train 8 and power train 9
v. Axis 5 *Requires* power train 8 and power train 9
vi. Axis 6 *Requires* power train 9

B.1.8.3 Example linear two-dimensional motion device

For the left motion device in Figure B.10 the *References* between axes and power trains are shown in Figure B.13.

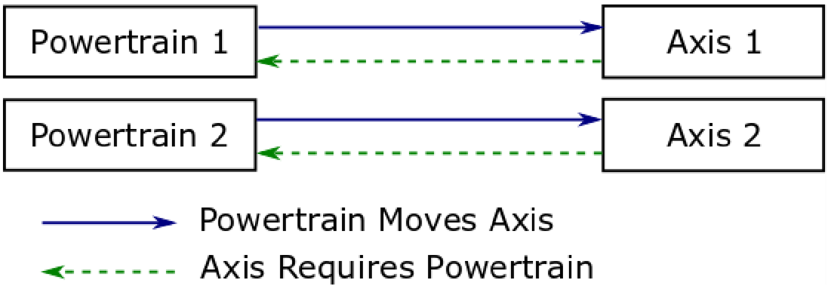


Figure B.13 – Coupling references for a simple linear two-dimensional motion device

Moves References:

iv. Power train 1 *Moves* axis 1
v. Power train 2 *Moves* axis 2

i. Axis 1 *IsMovedBy* power train 1
ii. Axis 2 *IsMovedBy* power train 2

Requires Refernces from power train to axis

i. Axis 1 *Requires* power train 1
ii. Axis 2 *Requires* power train 2

i. Power Train 1 *IsRequiredBy* axis 1
ii. Power Train 2 *IsRequiredBy* axis 2

For the right motion device in Figure B.10 the *References* between axes and power trains are shown in Figure B.14.

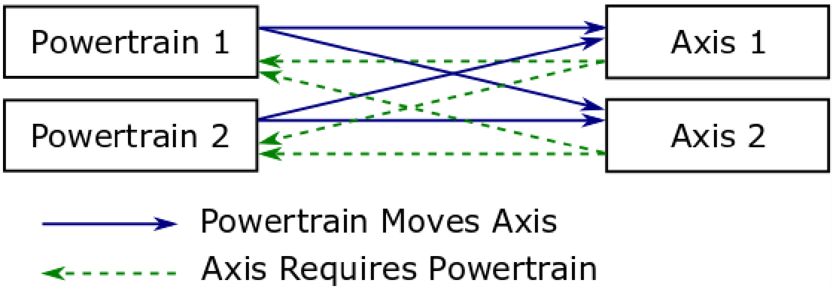


Figure B.14 – Coupling references for linear two-dimensional motion device

Moves References:

vi. Power train 1 *Moves* axis 1 and axis 2
vii. 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

Requires Refernces from power train to axis

iii. Axis 1 *Requires* power train 1 and power train 2
iv. Axis 2 *Requires* power train 1 and power train 2

iii. Power Train 1 *IsRequiredBy* axis 1 and axis 2
iv. Power Train 2 *IsRequiredBy* axis 1 and axis 2

B.1.9 Representations of exemplary server implementations

This chapter descripes different examples for the usage of DriveType or a SubType of ComponentType defined in OPC UA DI inclusive the references described in this specification.

All views show only the instances and references necessary to better illustrate the examples described.

B.1.9.1 ObjectTypes and references used with *DriveType* instances

Figure B.15 describes the usage of *DriveType* as an instance of a single-slot drive regarding the manipulator showed Figure B.10 on the left side.

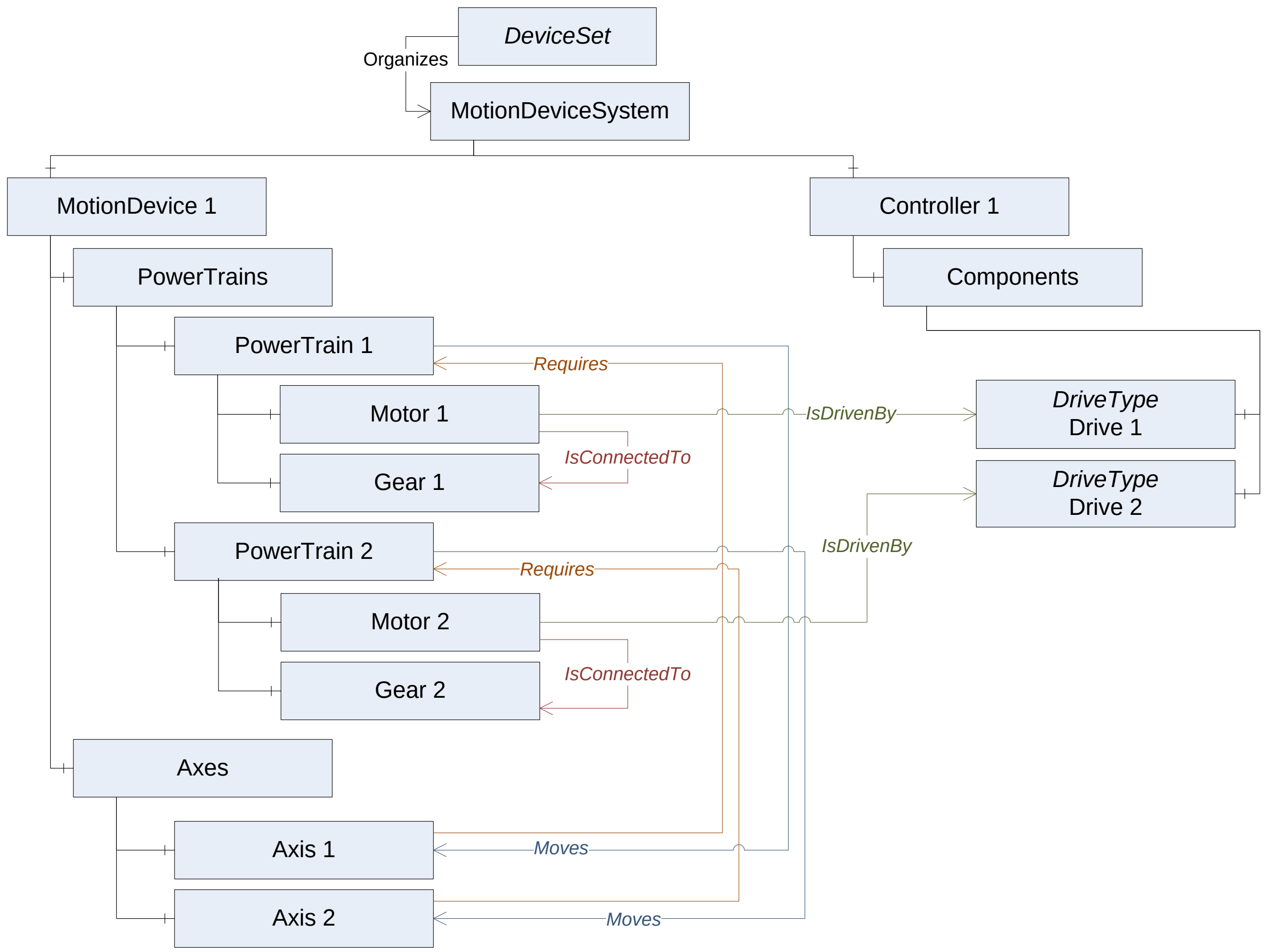


Figure B.15 – IsDrivenby references to DriveType instances

B.1.9.2 ObjectTypes and references used with instances of vendor specific subtypes of BaseObjectType for drive-channels

Figure B.16 describes the usage of slots or channels of a multi-slot-drive. The instance ot the slot is a vendor specific subtype of *BaseObjectType*.

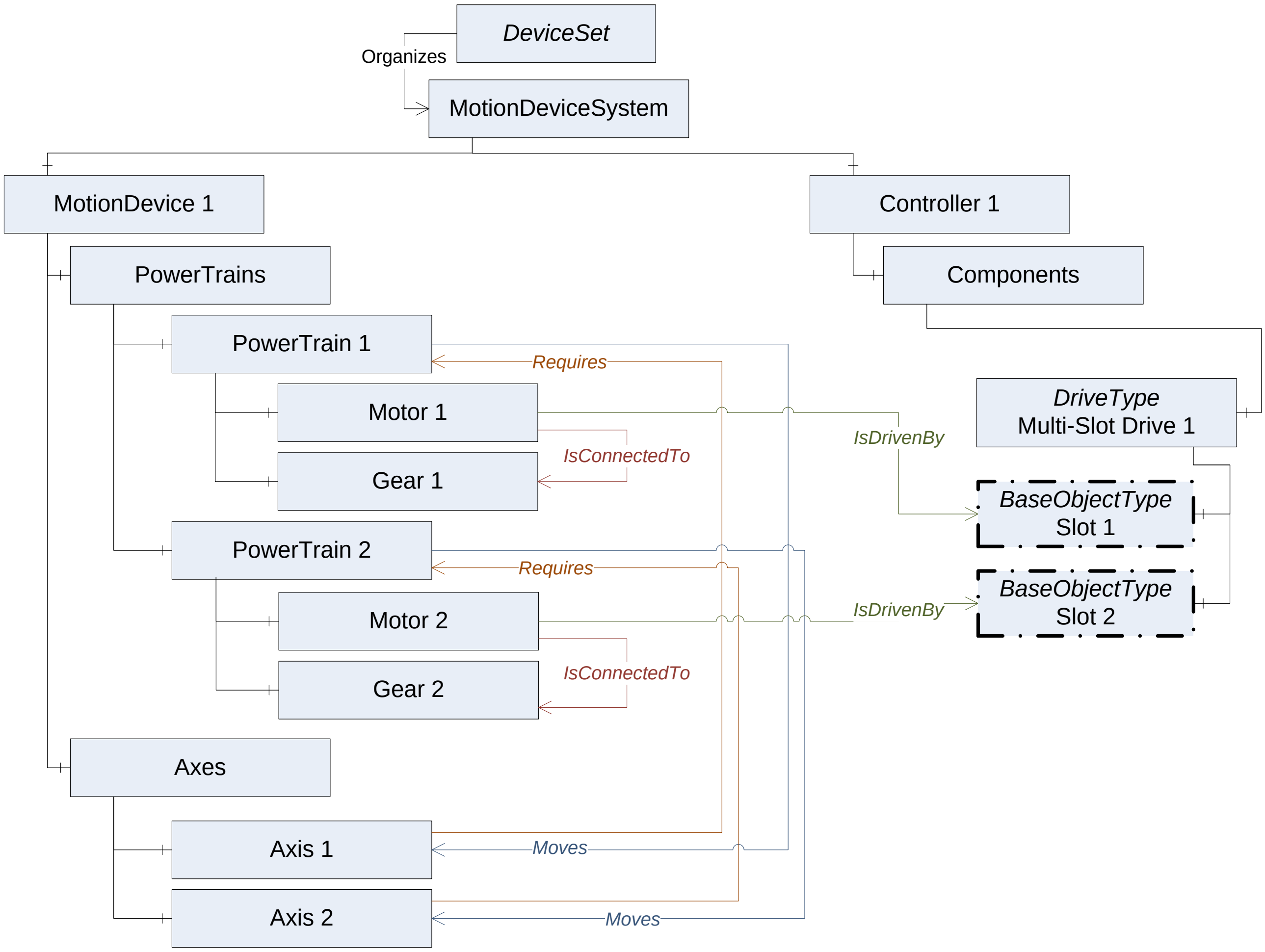


Figure B.16 – IsDrivenby references to vendor specific subtypes of BaseObjectType instances

B.1.9.3 ObjectTypes and references used with instances *DriveType* for drives with drive-channels

Figure B.17 describes the usage of *DriveType* for a multi-slot-drive if deeper information of slot definition is not available.

It is allowed that several instances of *MotorType* reference *IsDrivenBy* to one multi-slot-drive.

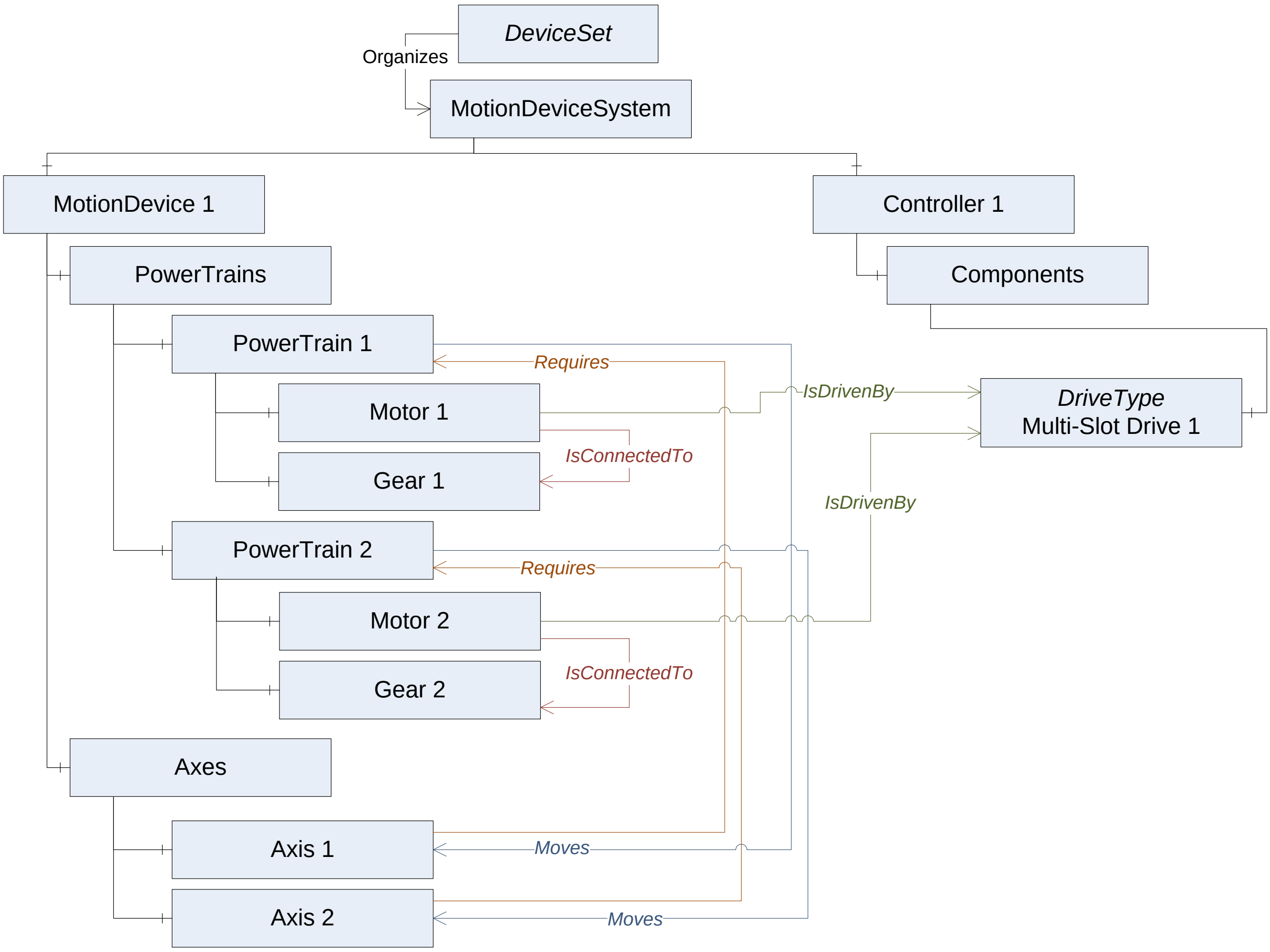


Figure B.17 – IsDrivenby references to DriveType instances for mulit-slot drives w/o slots

B.1.9.4 ObjectTypes and references used with instances of vendor specific subtypes of BaseObjectType for motor-integrated-drives

Figure B.18 describes the usage with a motor-integrated-drive as one physical device. The instance MyDrive is a vendor specific subtype of *BaseObjectType*. Identification properties of this physical device shall be defined within the referenced *MotorType*.

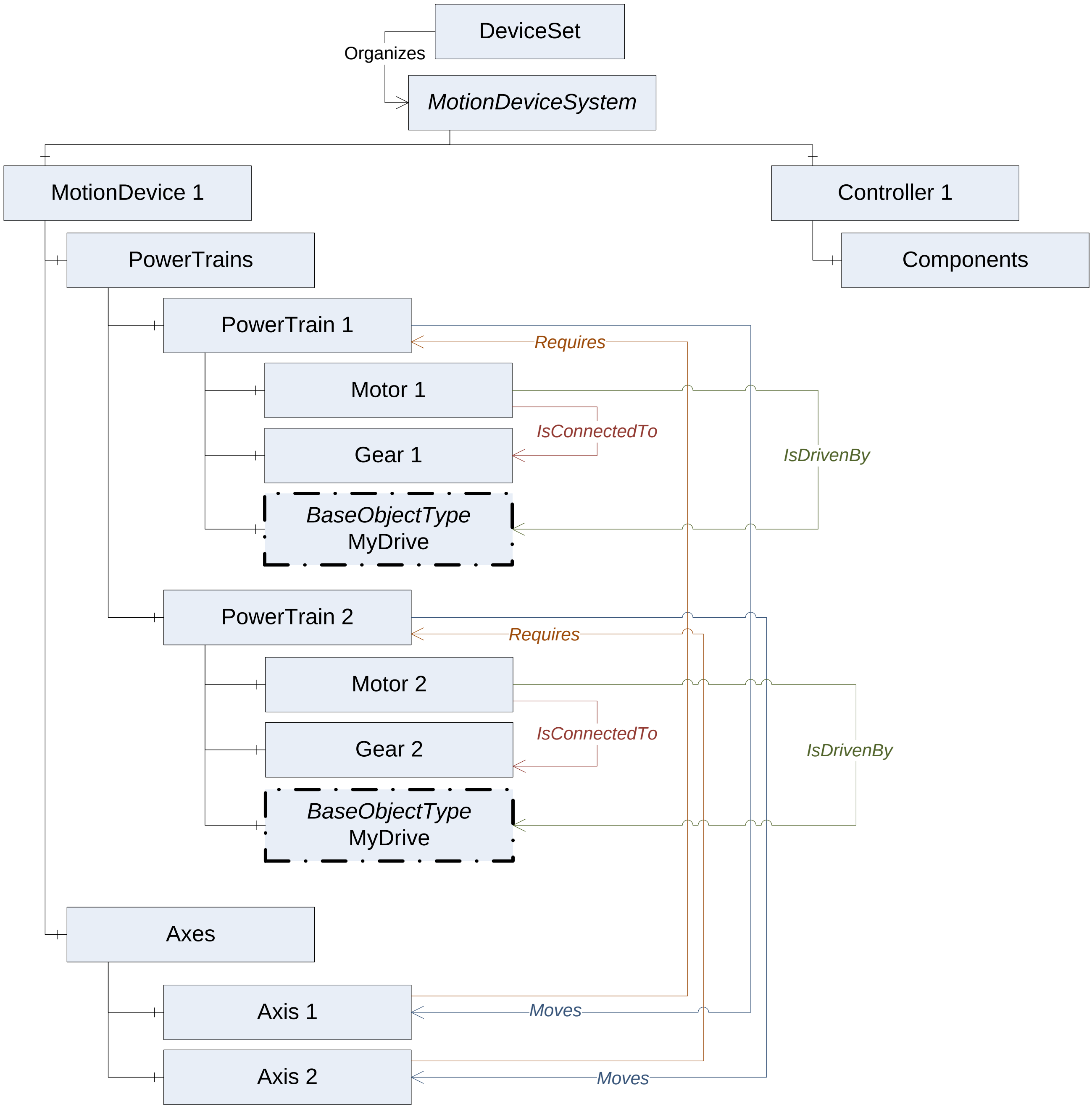


Figure B.18 – IsDrivenby used with motor-integrated-drives

B.1.9.5 Abstract example of a six-axis robot with master-slave axis and drive-slots

Figure B.19 describes an example view on a server with the instances of *ObjectTypes* and references of a six-axis robot with master-slave axis and drive-slots described in Annex B.1.8.2.

If a master-slave configuration only has one gear this shall be placed inside the master-power-train.

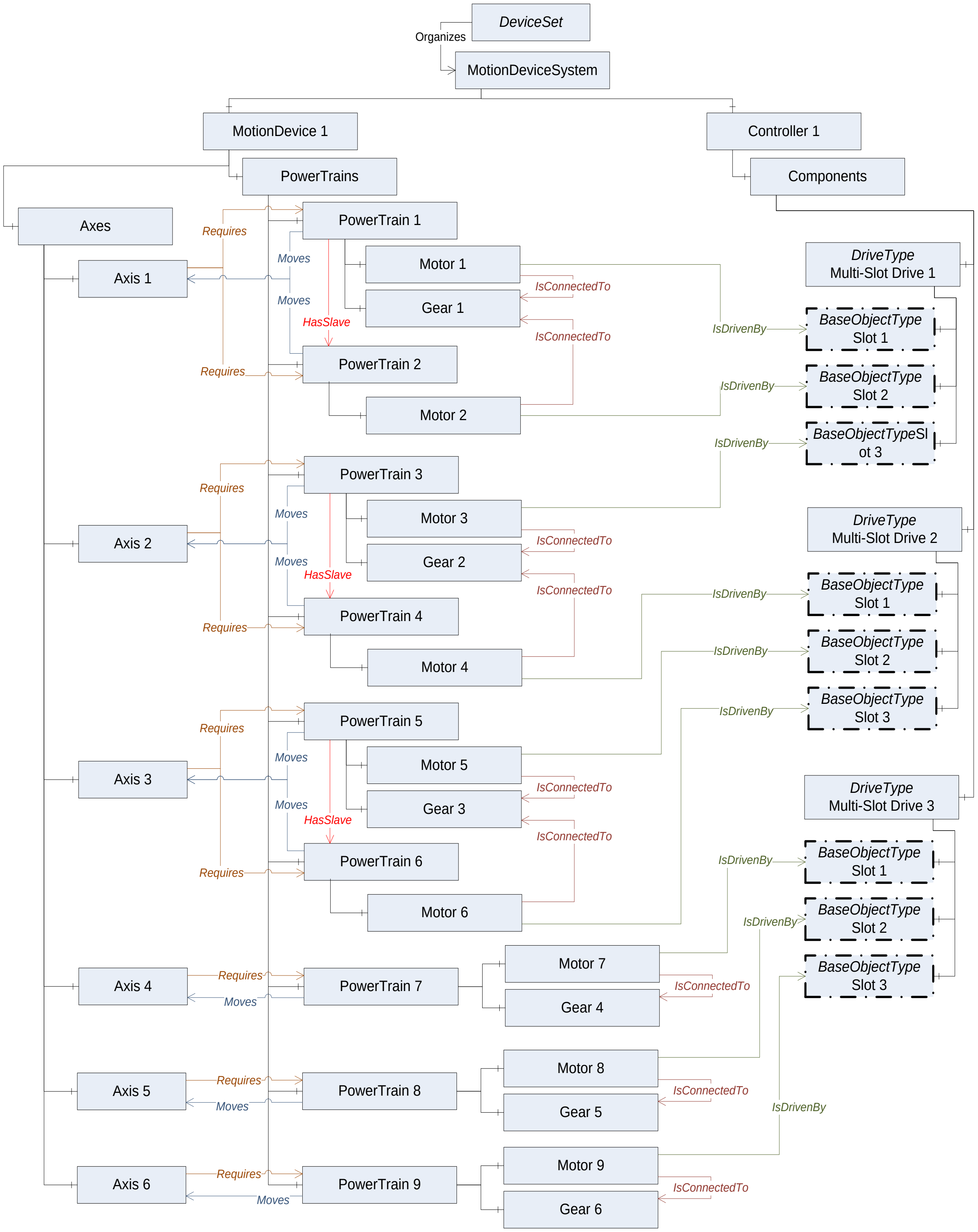


Figure B.19 – View on a six-axis robot with master-slave and drive-slots

B.1.9.6 Abstract example of a motion device system with three motion devices

Figure B.20 describes an example view on a server with the instances of *ObjectTypes* and references of a motion device system consisting of a six-axis robot, a linear unit and a turn-table which are controlled by one controller.

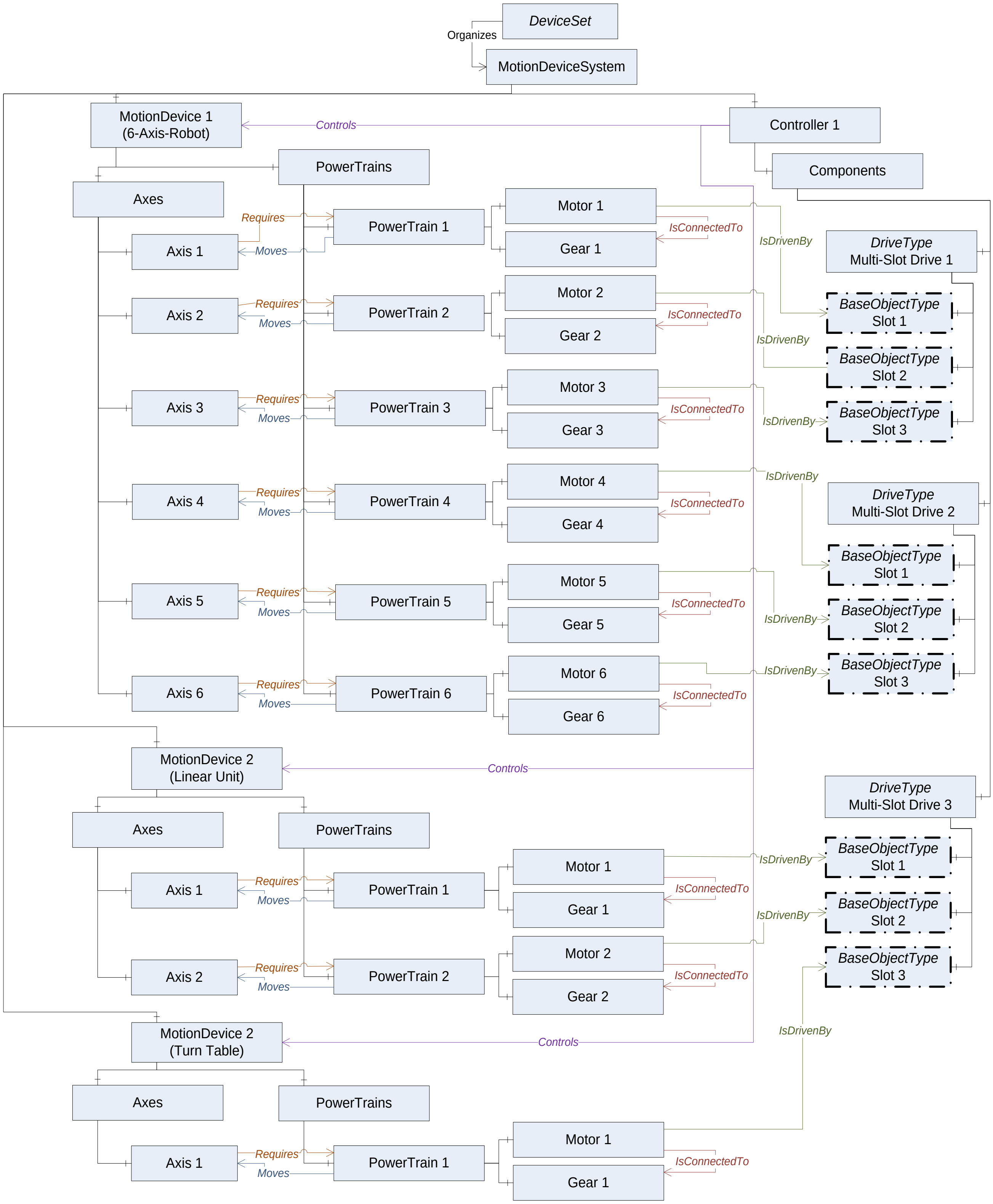


Figure B.20 – View on a motion device system with 3 motion devices controlled by one controller