FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Implementation of an OPC UA Server for a Robot Controller

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Hannes Brantner
Matrikelnummer 01614466

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Univ.Ass. Dipl.-Ing.(FH) Dieter Etz, MBA

Wien, 30. September 2019

Hannes Brantner	Wolfgang Kastner

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Implementation of an OPC UA Server for a Robot Controller

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Hannes Brantner
Registration Number 01614466

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Univ.Ass. Dipl.-Ing.(FH) Dieter Etz, MBA

Vienna, 30th September, 2019

Hannes Brantner

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Hannes Brantner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. September 2019

Hannes Brantner

Acknowledgements

I must thank Ao. Univ.-Prof. Dr. Wolfgang Kastner for handing me out a topic for the bachelor thesis immediately after asking him in the oral exam of Distributed Automation Systems. Thanks also to Dipl.-Ing. Thomas Frühwirth, who gave me an introduction to OPC UA and helped me when having problems in the lab. I also must thank ORev Heinz Deinhart, BSc who provided me with lab access and helped me with software license problems. Special thanks to Univ.Ass. Dipl.-Ing.(FH) Dieter Etz, MBA for his many wise tips and help regarding the implementation of the OPC UA Server and for establishing communication to Dr. Franz Eder who is the head of robotics and drive technology at GEVA Elektronik-Handels GmbH in Baden and provided us with many helpful manuals and documents to speed up the implementation process. I also want to thank Bernhard Müllner, BSc for providing support regarding the flashing and execution of robot programs. Last but not least, I have to thank my whole family for the fantastic support throughout my whole study time.

Abstract

As the OPC UA standard is quickly spreading in the field of industrial automation, the need arises to integrate existing devices that are non-compliant with this standard. The integrated device, in our case, was an industrial robot system that offered a telnet interface with proprietary commands. As the manipulation of proprietary robot controller software is tricky, an additional single-board computer was used as a gateway to host the OPC UA server and to wrap its services to proprietary telnet commands understood by the robot controller. If the developers use the right requisites, software for gateway devices can be implemented with reasonable effort. The advantages of complete interoperability are the rapid and painless exchange of information between all integrated devices and therefore, also between different levels of the automation pyramid.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Many industrial devices only offer proprietary communication interfaces. All these different interfaces lead to a complicated exchange of information and a lot of implementation work to do so. The high hurdle to exchange information can limit the performance of an entire plant because some essential communication channels might not get implemented due to high costs. Out of this dilemma arises the demand for a common standard, that unifies the spoken language between all devices in the automation pyramid. Accessible communication makes painless information exchange possible and helps to increase the utilization of many industrial devices using a higher amount of information provided. The ultimate goal is to achieve complete interoperability. The motivation came up to integrate the industrial robot system in our lab into the interoperable world of OPC UA to test the rising standard OPC UA in the interoperability domain on a small scale and check the feasibility to integrate devices initially non-compliant with this standard.

1.2 Problem statement

The initial problem was to write software for a single-board computer that acts as a gateway between the proprietary telnet commands of the robot controller and the high-level services offered by the OPC UA server which is also hosted on this single-board computer. At start-up, the single-board computer should wait for the availability of the robot controller's telnet interface and should then start offering its services. The software must at least support separate resuming and stopping of all robot programs currently loaded into the robot controller. Furthermore, there must be a way to ask the controller if it is currently idle or executing a robot program. All these features have to be provided using OPC UA functionalities via the hosted OPC UA server. Figure 1.1 shows a simple SysML block diagram of the target system.

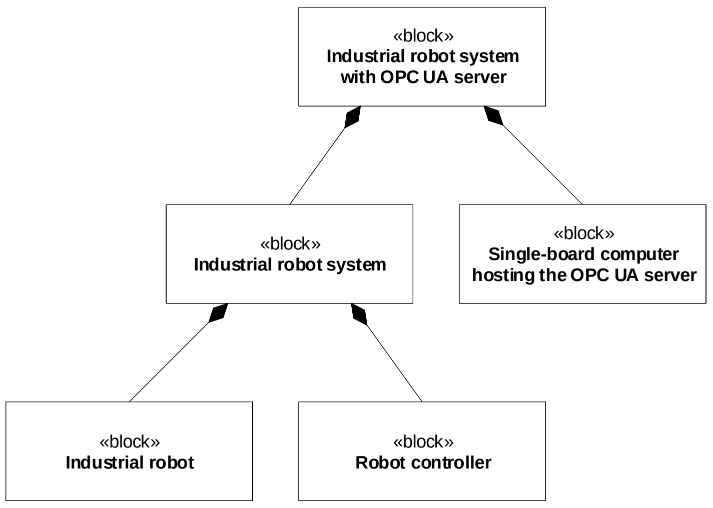


Figure 1.1: Block diagram of the target system

1.3 Aim of the work

The target of this thesis is to show the feasibility of integrating non-compliant devices into the OPC UA world with reasonable effort if the right requisites are used. This thesis should also show how the software used during the implementation process works and that the implementation of basic OPC UA features can be achieved with reasonable effort. Another aim is to demonstrate that legacy hardware can be made OPC UA compliant using gateway devices as described in this thesis. This option for retrofitting eliminates the need to buy new devices with OPC UA capability.

1.4 Methodological approach

The methodological approach is to study the manuals regarding the robot controller first. Then the telnet interface of the robot controller is going to be investigated, and many different commands with many different parameters should be issued to the controller to get a deep understanding, how this interface works. After this task, an OPC UA server framework should be chosen to speed up the implementation process and to enable working on a high abstraction level. This framework will also determine the programming language. The documentation regarding the OPC UA framework must be studied in-depth, and then the OPC UA services should be set up on the server and later called by an OPC UA client. The last task is to call the most suitable robot controller commands after activating the corresponding OPC UA service. The final application is then loaded to the single-board computer, and the whole system should then be tested extensively.

CHAPTER 2

Communication platform

2.1 Possibilities of machine-to-machine communication

The most common standards used for machine-to-machine communication are MQTT (Message Queuing Telemetry Transport), OPC UA, OPC Classic, and MTConnect (MT stands for manufacturing technology). The mentioned technologies are compared regarding interoperability, scalability, transportability, modeling, extensibility, conformity, and security. OPC UA and MQTT dominate the interoperability domain because they can be used on any system. OPC Classic is restricted to Windows and MTConnect is a read-only standard, meaning that it is only used to offer collected data from the shop floor via a RESTful interface. OPC UA and MQTT are also dominating the topic scalability because of the technologies support both simple and complex data structures. MQTT offers the highest data transmission rate. OPC UA dominates the modeling domain as the concept of a node guarantees extensibility and the feature to structure large amounts of data. OPC UA and OPC Classic products are tested by the OPC Foundation to ensure conformity with the standards. There is no such mandatory testing procedure for the two other standards. OPC UA is the only technology in the comparison that has also standardized the security aspects. As most of the points in the comparison are headed by OPC UA; this technology was chosen to work within this thesis. This paragraph is a summary of the comparison in [1, page 27-28]. Table 2.1 shows the used comparison table, which is a recreated and translated version of the original table from [1, page 27].

Property	OPC UA		OPC Classic		MTConnect		MQTT	
Interoperability	high due to base model	●	Windows only	◔	via adapters	◑	Pub/Sub	◕
Scalability	individual	●	single specification	◔	shop floor only	◑	almost arbitrary	●
Security	several concepts	●	individual	◑	HTTPS possible	◑	not in the protocol	○
Transportability	multiple technologies	◕	COM technology	◔	HTTP	◕	high performance	●
Modeling	complex models	●	single specification	◔	rigid model	◑	arbitrary structures	◕
Extensibility	Pub/Sub, discovery	●	pronounced	◑	agent function	◕	Pub/Sub	●
Conformity	multi-stage tests	●	multi-stage tests	◕	non-existent	◔	open standard	◑
Degree of Popularity	has risen in recent years	●	high in Europe	◕	primarily in the USA	◑	IoT	◕
Use in companies	clients and simple servers	◑	decreases due to OPC UA	◔	in the USA	◑	for sensors, simple devices	◑

Table 2.1: Comparison of machine-to-machine technologies [1, page 27]

2.2 Basics of OPC UA

2.2.1 OPC UA is not a protocol

It is a common misconception that OPC UA is just another communication protocol. Although OPC UA defines how the conversation between two parties starts and ends, and how to structure the message, it also systemizes how to model data, systems, machines, and entire plants. It is all about complete interoperability, and there is the ability to model everything. The technology allows to change the way data is organized, and information is presented. The most significant advantage of OPC UA as mentioned in [2, page 9] is that it organizes processes, systems, data, and information in a way that is unique to the experience of the industrial automation industry. UA is very scalable so that it can be used in embedded systems and large enterprise servers. The standard also supports state of the art security. It is the successor to OPC, which is now referred to as OPC Classic, but OPC UA outperforms the old standard in terms of platform independence, sophisticated data models, and security concerns. As mentioned in [2, page 26], OPC UA is still a developing technology, and changes will be made as feedback from adopters arrives. All the statements in this paragraph with no source specification can be found in [2, page 20-21].

2.2.2 Supported communication paradigms

An OPC UA server can be configured to serve an arbitrary number of clients. That is not true for many other industrial automation protocols like Modbus TCP, Modbus RTU, EtherNet/IP, and BACnet. Once a client/master takes ownership of the server/slave in these technologies, no other client or master can access it. This drawback is an advantage for OPC UA, which does not have this limitation. The core difference is that an OPC UA server can allow clients to discover the level of interoperability it supports dynamically. This level of interoperability means the discovery of services, the supported levels of security, and even type definitions for custom data and object types, for example. The above statements in this paragraph were taken from [2, page 22]. The newest version of OPC UA also supports a publisher-subscriber model for data exchange, which allows an encrypted one-to-many data transfer that was not possible beforehand. If a real-time capable technology as layer 2 of the OSI (Open Systems Interconnection) model is used, this data transfer can also be real-time capable and would stand in competition to many other factory floor protocols. Such a technology regarding layer two will be described in a section below. The claims about the publisher-subscriber model are taken from [3, page 66-67].

2.2.3 Platform-independent and scalable technology

The only requirements for OPC UA are the Ethernet capability and to know the current date and time. Current implementations of OPC UA are deployed on everything from small chips with less than 64K of code space to large workstations. As there are different layers of security available, one can choose an encoding mechanism that matches the hardware capabilities of the device. The scalability also applies to the address space of an OPC UA server. It can be comprised of a few objects with few variables or a complex set of interrelated objects in the case of a whole industrial plant. As OPC UA does not define expected behavior in the data link layer of the OSI model, new technology in this layer can be integrated seamlessly. OPC UA also integrates well with other IT systems because it supports the technologies SOAP and HTTP. SOAP describes the XML-based message format for communicating with Web Services. The term stood for "Simple Object Access Protocol", but it is no acronym anymore because SOAP 1.2 is not simple and not suitable to access objects. A SOAP message is an XML document, which has a root element called a SOAP envelope that specifies the version of the SOAP specification. It has up to two child elements called SOAP header and SOAP body. The SOAP header is optional, and its content is not standardized; in most cases, the header contains security-relevant information. The SOAP body is mandatory and contains the payload of the message, which is an XML document itself and not standardized. The structure of the XML document in the body must be known in advance by both communication partners. The statements regarding SOAP were taken from [4, page 84, 87-90]. The following listing [4, page 94] shows a necessary SOAP request that calls a procedure remotely by handing over the procedure name "bestellen" and the argument "1234":

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
    envelope">
    <env:Header>
        <t:transaction
            xmlns:t="http://thirdparty.example.org/
                Auftragsabwicklung"
            env:encodingStyle="http://example.com/encoding"
            env:mustUnderstand="true"/>
    </env:Header>
    <env:Body>
        <m:bestellen
            env:encodingStyle="http://www.w3.org/2003/05/soap-
                encoding"
            xmlns:m="http://ws.soa-buch.de/Annahme/">
                <m:nr>
                    1234
                </m:nr>
        </m:bestellen>
    </env:Body>
</env:Envelope>
```

Listing 2.1: Structure of a SOAP request

The response to this request would also be an XML-based SOAP message. Encoding in the OPC UA architecture is done using XML encoding or OPC UA Secure Conversion, which is an efficient, binary encoding for devices with limited resources. The intellectual property in the above sentences without source specification in this paragraph is taken from [2, page 22-23].

2.2.4 Address space model and information modeling

The fundamental building block of an OPC UA address space is a node. A node is described by its attributes/characteristics and interconnections to other nodes via references and relationships. All nodes share common attributes with other nodes in its NodeClass, where nodes are instantiations of its NodeClass. This concept is comparable to the object-oriented programming paradigm in the programming language domain. NodeClasses include the Variable NodeClass for defining variables, Reference NodeClass for defining references to other nodes and the Object Type NodeClass which provides type information for objects. A node can have up to twenty-two possible attributes. Some mandatory attributes are NodeClass, browse name, display name, and Node Id. An optional attribute would be the value attribute which is instantiated in nodes of the Variable NodeClass. The superior capabilities of nodes lead to the capability to form hierarchical relationships that represent systems, processes, and information. The sum of all the nodes is called the information model, which can be documented and communicated as an XML schema. It is the first technology of its kind to load, transport and references those information models in a running system. The above intellectual property was taken from [2, page 23-24].

2.2.5 Certifiable standard

There is a process to validate an OPC UA implementation, and there is an electronic test certificate being transmitted to the server in case of a successful compliance test. This certificate is documented evidence for the status of this server being a certified OPC UA device. Due to the scalable nature of OPC UA, different profiles exist that support different features and services. The Nano Profile is the profile supporting the least amount of functionality and services, and the Standard Profile supports the most amount of functionality and services. Each profile is composed of small sets of certifiable features called Conformance Units which are tested individually by the OPC UA Conformance Test Tool. This test tool also takes the supported transports and security profiles into account. If every Conformance Unit was tested successfully, the whole implementation is approved. The above statements in this paragraph were paraphrased from [2, page 25-26].

2.3 OPC UA Server

2.3.1 Basics of an OPC UA server

Just like other servers, OPC UA servers are endpoint devices that measure and digitize inputs and transform outputs to their analog equivalents. As mentioned above, an OPC UA server supports specific features of the OPC UA specification which are described as a profile. The profile also identifies the transport to be used, for example, HTTPS, HTTP, OPC UA Binary TCP or something else. The profile also determines the type and level of security supported by the server. Every server implements an address space which was also described above. Client requests are issued to the address space of the server or the server's OPC UA communication stack. The first type of request could be a read operation on a variable, a browse operation on the address space, adding or deletion of nodes or a request to get historical data and diagnostics. Keep in mind that only the services specified in the server's profile are supported. The other type of requests identify the type and level of security available or the supported transports. Supporting only one transport is enough for a server. This information is essential when creating new connections and sessions. All OPC UA servers support a discovery service that clients use to discover OPC UA servers that are available in the current network. When the discovery server is integrated into the same platform as the application server, it is called the Local Discovery Server (LDS). If the discovery server catalogs available servers in its address space, it is known as a Global Discovery Server (GDS). There is also an LDS with a multicast extension, which is called LDS-ME. The above paragraph was a summary of [2, page 27-35].

2.3.2 OPC UA server software architecture

The software architecture of an OPC UA server consists of the following three parts:

- System - The system provides the operating system and the transport layer for data communication.
- OPC UA software stack - This stack takes the client messages, for example, from the TCP/IP layer and tries to authenticate the client if a security strategy is active. The request is then decoded and further processed by so-called service sets. After processing is complete, a response is created, encoded, securitized, and then sent to the client. This part of the software architecture also maintains the Session management.
- User application space - The user application implements the information model and the OPC UA address space. This program code synchronizes real-world data to the OPC UA address space, and the real work of the application happens. Also, output data from the address space are written to the real world outputs. The user application space also processes method calls from the OPC UA server to start an electric motor, for example. The interface between the user application and the OPC UA software stack is called UA Server API.

The above paragraph was a summary of [2, page 30-32].

2.3.3 The Server object

The Server object is a node in a server's address space and is organized by the Objects folder object which is itself organized by the Root object which is the top-level object in an OPC UA address space. This object provides all the identification information of a device. This object is not mandatory for the low-end Nano Profile. It includes numerous variables like:

- Server Array - This variable is a set of pointers to remote OPC UA servers that are referenced in the server's address space.
- Namespace Array - This variable is a table of all URIs of namespaces used by nodes in the address space. Index 0 forms the OPC Foundation namespace, index 1 is the local OPC UA server namespace, and index two and above can be used for defining nodes of other organizations.
- Server Status - The Server status includes information like manufacturer, product codes, and software revision.
- Service Level - The quality of the service level is represented as a number from 0 to 255 (best) that clients can use to judge the relative reliability of servers in redundant server networks.

- Auditing - This variable is a Boolean indicating if the server is generating auditing events.

Some of the variables will be discussed further below. The server object also has four other important and mandatory objects as components:

- ServerCapabilities - This object contains a list of supported profiles, a list of signed software certificates from certification testing, the local IDs used for supporting multiple languages and many other variables. An OPC UA accesses this object first when connecting to an unfamiliar server to understand the supported functionality.
- ServerDiagnostics - This object contains variables like session count, view count, subscription counts, session timeout, and many more variables assisting the troubleshooting of the server.
- VendorServerInfo - This object exists to allow vendors to add additional proprietary information to the Server object.
- ServerRedundancy - This object describes the redundancy capabilities provided by the server.

The above paragraph was a summary of [2, page 32-33].

2.4 OPC UA Client

2.4.1 Basics of an OPC UA client

The controlling device in the OPC UA architecture is called the OPC UA client. A controller in almost all architectures sends outputs, receives inputs, and sends command requests to and from various devices. An OPC UA client furthermore supports the above described OPC UA discovery process to get the functionalities of a specific server, secure and authenticated connections, execution of service requests, and configuring notification send triggers from server to client. Keep in mind that there are no restrictions on how many clients can be connected to a single server or how many servers can be connected to a single client. The corresponding services for discovering servers are called Find Servers for LDS/GDS and Find Servers in Network for LDS-ME. The above statements were taken from [2, page 36-41].

2.4.2 How clients connect to and access servers

The Get Endpoints request is issued to the server which responses with an array of Endpoint Descriptions using the discovered or configured endpoint. An endpoint description consists of the supported transports, security mode, the server's application instance certificate, and the application description for the server. Then an appropriate endpoint is chosen. Knowing all relevant information, the client first must establish a connection to the server using the valid transport and security mode. Second, the client must establish a secure communication path between itself and the server. This so-called channel is a long-running, secure, and authenticated connection between the client and the server. The security key exchange takes place in this step. Certificates signed by Certification Authorities (CA) provide proof of the client's and server's identity. The last step is to establish a logical connection between the client application and the server application called a session. These sessions typically have a lifetime, and they must be renewed from time to time. Sessions do not need active underlying channels and are used for authorization. The server is allowed to reject the requests corresponding to step two and three. The above statements were taken from [2, page 39-40].

2.5 Address Space

2.5.1 Node

The address space in OPC UA is simple, incredibly flexible and can easily be stored as an XML file. Its base element is called a node which is a highly structured data element consisting of a set of predefined attributes and relationships/references to other nodes. Processes, systems, and information are represented as objects which are a collection of nodes. The intellectual property of this paragraph was taken from [2, page 64-69]. There are a total of eight NodeClasses, which are displayed in Figure 2.1.

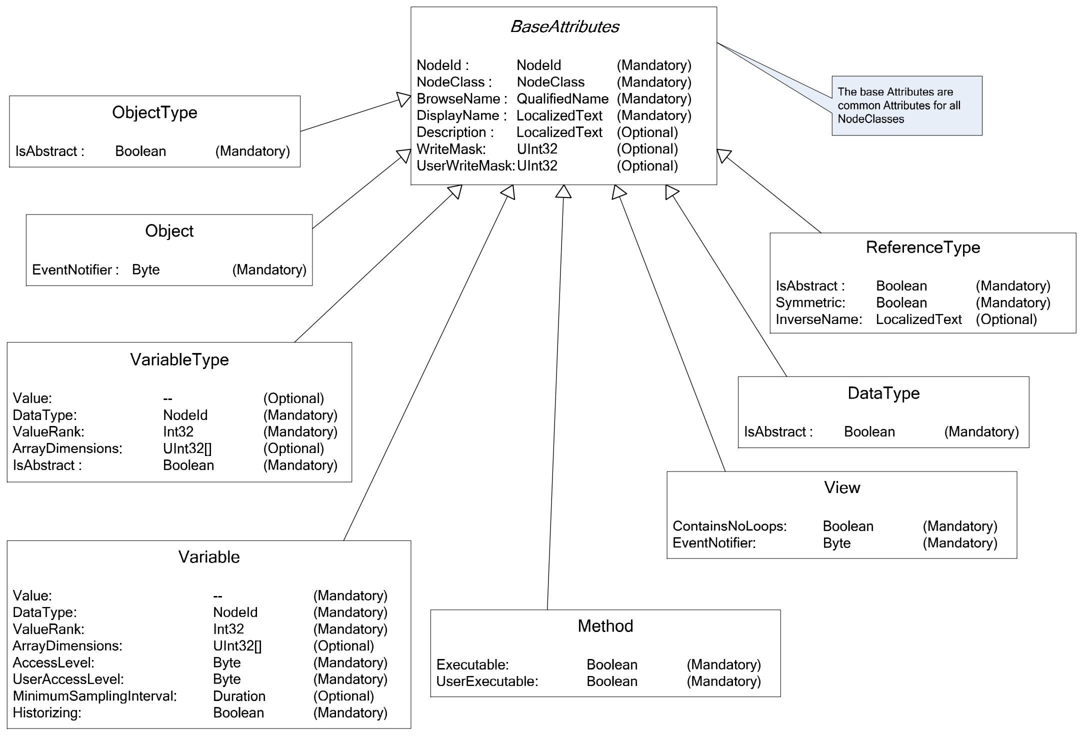

BaseAttributes
NodeId : NodeId (Mandatory)
NodeClass : NodeClass (Mandatory)
BrowseName : QualifiedName (Mandatory)
DisplayName : LocalizedText (Mandatory)
Description : LocalizedText (Optional)
WriteMask: UInt32 (Optional)
UserWriteMask: UInt32 (Optional)
The base Attributes are common Attributes for all NodeClasses
ObjectType
IsAbstract : Boolean (Mandatory)
Object
EventNotifier : Byte (Mandatory)
ReferenceType
IsAbstract : Boolean (Mandatory)
Symmetric: Boolean (Mandatory)
InverseName: LocalizedText (Optional)
VariableType
Value: -- (Optional)
DataType: NodeId (Mandatory)
ValueRank: Int32 (Mandatory)
ArrayDimensions: UInt32[] (Optional)
IsAbstract : Boolean (Mandatory)
DataType
IsAbstract : Boolean (Mandatory)
View
ContainsNoLoops: Boolean (Mandatory)
EventNotifier: Byte (Mandatory)
Variable
Value: -- (Mandatory)
DataType: NodeId (Mandatory)
ValueRank: Int32 (Mandatory)
ArrayDimensions: UInt32[] (Optional)
AccessLevel: Byte (Mandatory)
UserAccessLevel: Byte (Mandatory)
MinimumSamplingInterval: Duration (Optional)
Historizing: Boolean (Mandatory)
Method
Executable: Boolean (Mandatory)
UserExecutable: Boolean (Mandatory)

Figure 2.1: A class diagram of all eight NodeClasses and their attributes [5, page 333]

The three most important NodeClasses are the Object, the Variable, and the Method NodeClass. These concepts are known from the object-oriented programming paradigm. Objects can fire events and have methods and variables. They are used for organizing and could represent a real-world machine, process, system, or component of it. The root node is also a node of the Object NodeClass. Nodes of the Variable NodeClass are the only nodes in OPC UA that mandatorily contain data values. Nodes of the NodeClass Method represent a method that can be called by the client. The input arguments the client has to provide, and output arguments the client can expect are specified in the Method node. The above statements were taken from [5, page 30-31]. Views and events will not be discussed in this thesis, as they were not used in the implementation.

2.5.2 Attributes

Every node of an OPC UA NodeClass is described by some of the twenty-two predefined attributes. These attributes can be mandatory, optional, or disallowed for a specific NodeClass. The figure above shows the available attributes for every NodeClass. The four mandatory attributes used by all NodeClasses are the NodeId, the NodeClass, the BrowseName, and the DisplayName. These attributes are displayed as properties of the BaseAttributes class, which is only a class in the diagram and no NodeClass in the OPC

UA architecture. The above statements in this paragraph were taken from [2, page 65-71]. A short description of all attributes can be found in Table 2.2.

Attribute	ID	Description
NodeId	1	The server unique identifier for the node
NodeClass	2	The base type of the node
BrowseName	3	A nonlocalized, human readable name for the node
DisplayName	4	A localized, human readable name for the node
Description	5	A localized description for the node
WriteMask	6	Indicates which attributes are writeable
UserWriteMask	7	Indicates which attributes are writeable by the current user
IsAbstract	8	Indicates that a type node may not be instantiated
Symmetric	9	Indicates that forward and inverse references have the same meaning
InverseName	10	The browse name for an inverse reference
ContainsNoLoops	11	Indicates that following forward references within a view will not cause a loop
EventNotifier	12	Indicates that the node can be used to subscribe to events
Value	13	The value of a variable
DataType	14	The node id of the data type for the variable value
ValueRank	15	The number of dimensions in the value
ArrayDimensions	16	The length for each dimension of an array value
AccessLevel	17	How a variable value may be accessed
UserAccessLevel	18	How a variable value may be accessed after taking the user's access rights into account
MinimumSamplingInterval	19	Specifies (in milliseconds) how fast the server can reasonably sample the value for changes
Historizing	20	Specifies whether the server is actively collecting historical data for the variable
Executable	21	Whether the method can be called
UserExecutable	22	Whether the method can be called by the current user

Table 2.2: List of all attributes with a short description [5, page 334]

2.5.3 References

References are links that relate nodes to one another. A reference is of a specific type and identifies both the source and the target node. These types are specified via nodes of the NodeClass ReferenceType. There are hierarchical reference types like HasComponent, Organizes, and HasProperty, but also non-hierarchical reference types like HasTypeDefinition. The reference type HasTypeDefinition is used to link nodes of the Variable and Object NodeClass to their corresponding definition nodes of the NodeClasses VariableType and ObjectType. The above information was taken from [2, page 65]. References are also distinguished between symmetric and nonsymmetric references. A nonsymmetric reference would be the reference type HasEncoding, which is only valid in the direction of the reference. For example, the custom reference type IsSiblingOf would be symmetric because the reference is valid from source to target an vice versa. The information about reference symmetry is taken from [5, page 23-24]. The hierarchy of all base reference types is shown in Figure 2.2.

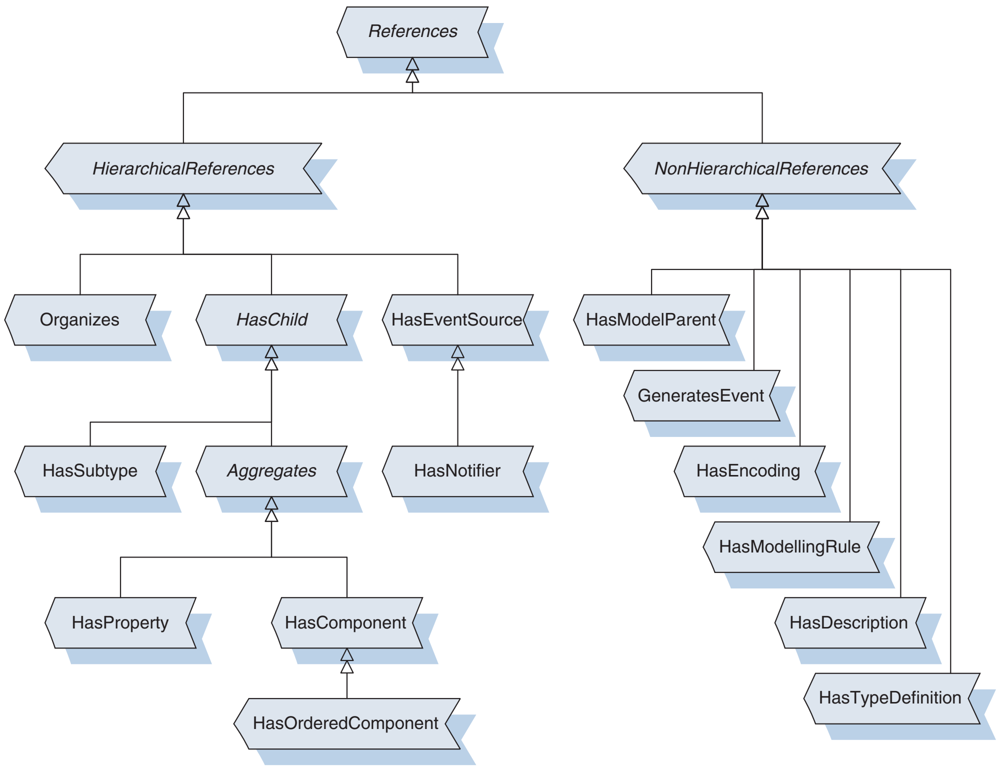


Figure 2.2: Hierarchy of all base reference types [5, page 335]

2.6 Information Modeling

OPC UA has standardized the documentation, implementation, reference, and access to information models. An information model is a logical representation applied to a physical process. The concept of a node allows the modeling of real-world entities as objects that have variables and methods, as mentioned before. Which real-world parameters are modeled as variables, which real-world functionalities are modeled as methods and how the references of the modeled nodes are created and their attributes are filled is up to the designer of the information model. There are lots of different possibilities to model real-world entities in information models. Therefore there will always be differences between two independently created information models of these entities. These differences in modeling would end up in repeating the integration process and would, later on, lead to interfacing problems. Standardized information models called OPC UA companion specifications were introduced, which will be described later, to avoid these problems. OPC UA Information Modeling is different from other technologies in many ways:

- Consistent structure and standardized definitions
- Consistent and standard structure to the documentation of the model (XML)
- A mechanism for translating the information model into a real-time address space
- Standard mechanism for clients to identify the model and access component definitions and type information at run time
- The encoding, securing and transporting of values in the real-time address space are independent of the information model development

For type definitions common to multiple installations or throughout an industry domain, the type definitions can be located remotely. In this case, the type definition reference includes a common URI (Uniform Resource Identifier) used for all definitions of that type. This URI can be converted with the help of the server's Namespace Array to the namespace index. This index is a component of the NodeID, which is an attribute every node has. Clients need the namespace index to know, where to look up type definition, for example. This whole topic is critical to trade associations, which understand that integration costs can be dramatically lowered if everyone uses a standard definition for entities. Most of the system problems lie in the interfacing between vendors, and these problems often arise at the worst possible time: deployment and start-up. Standardized information models that are implemented by an OPC UA server could solve this problem with ease. Most information models are currently created using advanced GUI tools. There is a difference between the address space and the information model. However, both use the same organization based on nodes. The information model specifies the high-level organization of an entity. It consists most often of nodes defined in OPC UA companion specifications and vendor-specific nodes. The address space model describes the specifics and how that model is deployed in an OPC UA server device. The specifics of the address space are mapped to the real world by accessing sensors and actuators. The above statements were taken from [2, page 76-87]. Figure 2.3 shows the overall infrastructure. The red wavy lines under the word "PLCopen" have been removed from the original image, and the resulting image has been vectorized.

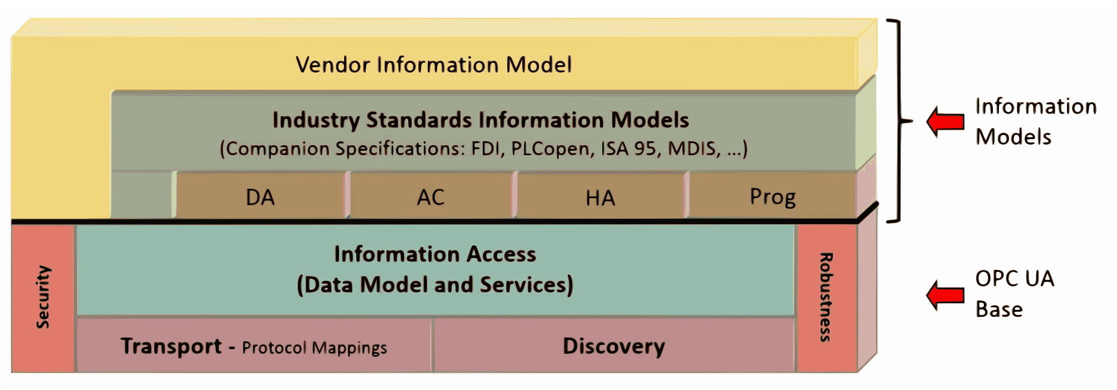

Vendor Information Model
Industry Standards Information Models
(Companion Specifications: FDI, PLCopen, ISA 95, MDIS, ...)
DA
AC
HA
Prog
Information Access
(Data Model and Services)
Security
Robustness
Transport - Protocol Mappings
Discovery
Information Models
OPC UA Base

Figure 2.3: OPC UA Base Services Architecture [6]

2.7 Data Typing

OPC UA is a strongly-typed technology, and there are four kinds of DataTypes: Built-In DataTypes, Simple DataTypes, Enumeration DataTypes, and Structured DataTypes. All DataTypes are modeled as nodes of NodeClass DataType. All attributes of a node have a fixed data type excluding the Value attribute. The ValueRank attribute of the NodeClasses Variable and VariableType determines if the Value is scalar, an array or any of these two and the array dimensions can optionally be specified with the ArrayDimensions attribute. Built-in DataTypes are a predefined set of DataTypes defined by the OPC UA specification that all clients and servers must inherently understand. They cannot be extended by vendor-specific or standardized information models. Some representatives of this kind would be Int32, Boolean, Double, NodeId, LocalizedText, and QualifiedName. The following three kinds of DataTypes can be extended with custom DataTypes by information models. Clients can identify and match types from different servers as identical types by observing origins and derivations [2, page 89]. Simple DataTypes are subtypes of Built-In DataTypes. The concrete data values cannot be distinguished between the actual type and supertype. The representation on the wire is also the same. Simple DataTypes should add more semantics to a data variable. An example would be the Duration DataType, which is a subtype of Double defining a time interval in milliseconds. Enumeration DataTypes represent a discrete set of named values and are represented as the Built-In DataType Int32 on the wire. The mapping from integer to named values is implemented by using a property called EnumStrings. EnumStrings is an array of LocalizedText that allows the mentioned mapping. Structured DataTypes represent structured data. They are a struct consisting of several DataTypes. An example would be the DataType Argument, which is used to define an argument of a Method. It contains the description, data type, and name of the modeled argument. There are also Abstract DataTypes like Number and BaseDataType, which are only used to structure the DataType hierarchy. The node BaseDataType specifies the root of this hierarchy. The above statements without source specification were taken from [5, page

61-71]. Figure 2.4 shows the hierarchy of DataTypes in OPC UA.

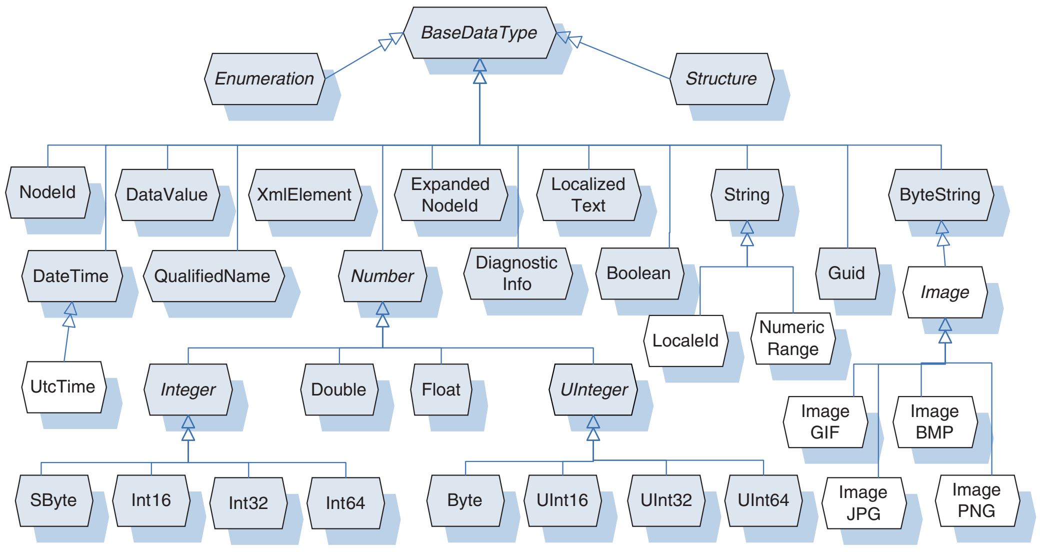


Figure 2.4: OPC UA DataType hierarchy [5, page 335]

2.8 OPC UA Companion Specifications

OPC UA companion specifications are information models that address dedicated industry problems and are defined by specific working groups. These working groups consist of members of the OPC Foundation and members of another organization, which must be a relevant one in the specific industry domain. To yield a high acceptance of the defined information model, the OPC Foundation cooperates with several organizations like ISA[1], MIMOSA[2], VDMA[3] or OMAC[4]. These organizations can harmonize the information model with their standards. All published companion specifications are available as nodeset files following the links on [7]. Companion specifications enable interoperability at the semantic level and speed up the implementation process. There are three different types of companion specifications:

- Internal - OPC-internal working groups create these information models.
- Joint - These are models created in a joint working group between the OPC Foundation and another organization.
- External - These are models created independent of the OPC Foundation.

[1]International Society of Automation
[2]Machinery Information Management Open Systems Alliance
[3]Verband Deutscher Maschinen- und Anlagenbau
[4]Organization for Machine Automation and Control

This list can also be found on [7]. The other information in this paragraph was taken from [3, page 27].

2.8.1 OPC UA Standard Model

This standard information model defines all Built-In DataTypes and is furthermore known as the base model, from which all other models derive. It was only designed by the OPC Foundation and can be labeled as an internal companion specification. The standard model is the agglomeration of the various standardized single models listed on the website [8]. For example, the Boolean, Double, NodeID, and Byte data types are defined here. The model is hosted on `https://github.com/OPCFoundation/UA-Nodeset/blob/master/Schema/Opc.Ua.NodeSet2.xml`.

2.8.2 OPC UA Devices

This internal companion specification builds upon the standard model and defines additional types used to model devices. It defines the TopologyElementType, which is the base object type for elements in a device topology. The TopologyElementType has a ParameterSet to model all parameters of a device and a MethodSet to list all supported method calls. The defined ComponentType is a subtype of the TopologyElementType, and the DeviceType of this model is a subtype of the ComponentType. The DeviceType is typically used to model field devices. It also models the DeviceSet, which is an instance of the BaseObjectType and is organized in the Objects folder. All devices in this model must be instantiated under the DeviceSet using HasComponent references to locate the devices in the address space quickly. The above information was taken from [9, page 22-23, 37-38]. It is hosted on `https://github.com/OPCFoundation/UA-Nodeset/blob/master/DI/Opc.Ua.Di.NodeSet2.xml`.

2.8.3 OPC UA for Robotics

This information model was created by a joint working group of the OPC Foundation and VDMA, so it is called a joint companion specification. It builds upon the standard model and the device model, which are both described above. The model defines the MotionDeviceSystemType as a subtype of the ComponentType from the OPC UA Devices model. This newly defined type should be instantiated in the DeviceSet to model a real-world robot system. The MotionDeviceSystemType consists of three components: MotionDevices, Controllers, and SafetyStates. These nodes should consist of as many MotionDeviceType, ControllerType, and SafetyStateType nodes as applicable to the real world robot system. The above intellectual property was taken from [10, page 27]. Maybe this model will also be hosted on Github in the future.

2.9 OPC UA over TSN

Time-Sensitive Networking (TSN) can be used on the field level of the automation pyramid because it is real-time capable. This feature is achieved by adding real-time extension standards to the Ethernet standard, which belongs to layer 2 of the OSI model. These extensions address the following problems in the time-sensitive traffic domain: time synchronization, bounded latency, reliability, and resource management:

- Time synchronization must maintain a common notion of time across all links even in case of failure of a link or the grandmaster. This behavior is described in the two standards, IEEE 802.1AS-2011 and IEEE 802.1AS-Rev. Implementations of the first standard have demonstrated to achieve a reference time with an accuracy better than one microsecond. The standard IEEE 802.1AS-Rev has improved redundancy compared to the IEEE 802.1AS-2011 standard and also fixes the time jump issue of the IEEE 802.1AS-2011 standard if a new grandmaster in case of failure is elected. The intellectual property in this bullet point was taken from [11, page 1097-1098].
- Bounded low latency is an essential goal for messages with real-time constraints. There are currently three standards in the TSN domain that are trying to achieve this requirement: IEEE 802.1Qbv, IEEE 802.1Qbu, and IEEE 802.1Qcr. The standard IEEE 802.1Qbv defines schedule-driven communication that uses the synchronized time in transmission and forwarding decisions for messages in the network. The next standard IEEE 802.1Qbu specifies a preemption algorithm that allows time-critical messages to interrupt ongoing non-time-critical transmissions. The last standard IEEE 802.1Qcr standardizes asynchronous traffic shaping that uses an urgency-based scheduler. The intellectual property in this bullet point was taken from [11, page 1099].
- High reliability is another essential requirement because most TSN applications cannot tolerate the delay due to retransmissions of lost frames. The standard IEEE 802.1CB achieves high reliability by specifying redundant packet transmission over separate paths through the network and the elimination of duplicated packets at or nearby the destination. Another standard that tries to achieve high reliability is IEEE 802.1Qca. This standard offers precise path control and integrates a tool for bandwidth and stream reservation along the forwarding path and resiliency control mechanisms for data traffic. The last standard regarding reliability is IEEE 802.1Qci. It defines procedures to make filtering decisions and enforce policing on a per-stream basis. The standard also offers a quality of service protection to streams by taking mitigating actions to non-conforming streams. The intellectual property in this bullet point was taken from [11, page 1099].
- Resource management is needed to achieve deterministic networking. Bandwidth reservation is made by establishing and enforcing a bandwidth contract between the network and the application. This contract limits the source of a TSN flow to

comply with a maximum packet size and the number of packets transmitted per time intervals. All needed network resources can, therefore, be reserved for specific traffic streams traversing a bridged local area network. This task is achieved by the stream reservation protocol (SRP) in the IEEE 802.1Qat standard and its enhancements in the IEEE 802.1Qcc standard. Another standard in this domain is IEEE 802.1CS, which defines procedures to replicate an extensive registration database from one end to the other of a point-to-point link. The intellectual property in this bullet point was taken from [11, page 1099-1100].

TSN also addresses the demand for high bandwidth in some application areas, a high level of security, and a high level of interoperability. Therefore, TSN can transport all the data in and between all the different layers of the automation pyramid. The aim of OPC UA over TSN is the establishment of an open, uniform, and compatible Industrial Internet of Things (IIoT) solution for real-time capable peer-to-peer communication. TSN is responsible for synchronization and the deterministic delivery of data packets. OPC UA is responsible for a standard format of the application data to be understood by the sender and the receiver. When comparing this situation to a phone call, TSN ensures a proper connection quality and OPC UA ensures the same language of the participants so that they can communicate in real-time. This combination could replace all other technologies for exchanging information in the automation pyramid over a single network. This standard is suitable for not time-critical report data, but also suitable for time-critical data packets between control units, for example. The statements without source specification in this paragraph were taken from [3, page 181-184].

2.10 Comparison of existing approaches to migrating non-compliant devices

The industrial engineering is focused on maintaining current installations and on installing device upgrades only from time to time [3, page 79]. There are two scenarios when migrating to IIoT technologies. The first scenario is called the "Brownfield" installation and deals with legacy equipment and legacy software that performs discrete functions in isolation. The first step of the migration process is called "Retrofitting" and involves adding sensors to legacy hardware if necessary to gain more relevant information about the industrial process. The second step is called "Bridging the gap", and it involves adding a gateway device that can acquire and offer the process data on the Internet as a server. The second scenario is called the "Greenfield" installation and indicates a situation where no preexisting equipment is present. This situation allows the installation of smart cyber-physical systems, which are interoperable with each other. Open communication protocols and open standards like OPC UA are used to satisfy complete interoperability. The above statements are taken from [12, page 3]. To utilize all the advantages of OPC UA like plug-and-play functionality and efficient and reliable data communication, one must migrate the current installation to be compatible with OPC UA. This necessary migration can be achieved using the following approaches:

2.10.1 Rapid mass migration

When using this approach, the OPC UA compatibility of all devices is implemented at once. Every device can reference every other device in this consistent system, and this gives an immediate advantage. In the long term, system maintenance will be simplified as a reduction of proprietary protocols and devices takes place. If a device must be replaced, there is no compulsion to repurchase it at the same manufacturer. Problems with unreliable data communication between heterogeneous devices belong to the past. The bigger the complexity of the industrial automation plant, the more complex the rapid mass migration approach will get. This approach is more feasible for smaller industrial companies, where plant complexity is manageable. Also, total migration costs must be considered in the decision-making process. This paragraph summarizes the statements from [3, page 80-81].

2.10.2 Slow migration with support for multiple legacy protocols

This method is the most straightforward approach when migrating to OPC UA because it takes little time, effort, and costs in the beginning. Every time an old device is replaced or upgraded, it must provide OPC UA functionality afterward. This process can proceed very slowly, but eventually, in the future, all devices in an industrial plant will support OPC UA. The disadvantage of using this approach is that many different protocols are used during the migration period. This drawback leads to an increasing effort in the configuration of software in the automation pyramid and increasing operating costs. This step-by-step approach is more feasible for big industrial plants but can take more than a decade. This paragraph summarizes the statements from [3, page 81-82].

2.10.3 Migration via OPC UA gateways

This approach is like our approach to the migration problem of the industrial robot system. Devices called OPC UA gateways convert proprietary protocols and expose their data using an OPC UA server to the clients. These OPC UA clients will mostly be applications in the MES and SCADA layer of the automation pyramid. The communication of the lower levels of the automation pyramid can take place using proprietary protocols as before and is not interfered with using this approach. Legacy devices can be upgraded with OPC UA functionality by this approach, and rapid migration is also possible in this case. This paragraph summarizes the statements from [3, page 82-84].

CHAPTER 3

Explanation of software, models, and interfaces

3.1 Programming language

The high-level programming language Python was used to implement the OPC UA server because the problem statement is not critical regarding time and operating on a high-level was never a bad idea when looking back in computer science history. Python offers great language constructs to ease and speed up the development process like string slicing, method references, context managers, easy-to-use packages distributed via pip, and excellent error handling. For embedded devices, Python is well-suited because python files are compiled to platform-independent bytecode and then interpreted directly and do not need an entire compilation into machine code to run correctly, so fewer compatibility problems arise. All the programming language's mentioned, and unmentioned features can be found on [13]. For this project, the chosen Python implementation was CPython, which is hosted on `https://github.com/python/cpython`. The source code was written using the Pycharm Python IDE developed by JetBrains because of the code analyzer, the debugger, and the run configurations. All features of Pycharm can be found on [14]. Git was used to tracking the source code changes during the developing process. All features of Git can be found on [15].

3.2 OPC UA server framework methods

Most of the time, an OPC UA server is implemented, this is done with the help of frameworks. The library python-opcua was used, which is hosted on `https://github.com/FreeOpcUa/python-opcua` and distributed as the package opcua on pip. This library can be used for the implementation of OPC UA clients and servers. The framework can start and stop servers and set up clients. Nodes of the server's address space can be created/modified, and an XML importer/exporter is also present on the server-side. Methods imported from XML nodeset files are linked to Python method references using a framework function, and the documentation is entirely proper. The features mentioned above, the methods/functions used and described below and lots of other features can be found on the documentation website [16]. Code samples are listed in the Appendix chapter.

3.2.1 Server(shelffile=None, iserver=None)

This constructor creates an OPC UA server instance with default values. During start-up, the initial address space is created using the standard model, which was described above. A call to this method can be very time consuming on less powerful devices. Therefore, a cache file path can be passed using the shelffile parameter, where a cache file will be created to speed up future start-ups. The iserver parameter is a reference to an InternalServer object if an existing object should be used. All methods below are methods of the Server class:

set_endpoint(url)

This method sets the endpoint, where the OPC UA server exposes its services.

set_security_policy(security_policy)

This method is used to set the supported security policy of the server. The argument is a list of security policy type values taken from the enumeration ua.SecurityPolicyType. If the list contains an actual security policy type that uses encryption, a private key and a certificate must be set.

set_server_name(name)

This method sets the name of the server, that is going to be displayed in the OPC UA client.

import_xml(path=None, xmlstring=None)

This method imports nodes defined in information models, which are saved in XML format, into the server address space. A path to the XML file or a string with the file content can be passed via the parameter xmlstring.

get_node(nodeid)

This method returns a node from the server's address space that matches the NodeId object or a string representing a NodeId.

link_method(node, callback)

This method links a python function using the callback parameter to a node object representing an OPC UA method.

start()

When calling this method, the server starts to listen to the network and will serve clients.

3.2.2 Node(server, nodeid)

This constructor returns a node from the server's address space that matches the NodeId object or a string representing a NodeId. The Node constructor is called when executing the server method get_node(nodeid). All methods below are methods of the NodeClass:

delete(delete_references=True, recursive=False)

This method deletes a node from the address space. The parameter delete_references specifies if the references should also be deleted and the parameter recursive specifies if also all nodes should be deleted that have hierarchical references pointing up to that node that is going to be deleted. Therefore, for example, all components of the deleted node will be deleted, too.

add_object(nodeid, bname, objecttype=None)

This method adds an instantiated object type to the calling node using the HasComponent reference. The object type can be specified using a NodeId string or NodeId object. Furthermore, a NodeId string or NodeId object for the new node must be specified. Keep in mind that nodes created by the server should reside in the namespace with index 0. The bname parameter specifies the browse name of the node that is going to be created.

get_child(path)

This method follows the path to a child node starting by the calling node. The path can be a list of strings that represent browse names of the child nodes.

set_data_value(value, varianttype=None)

If the calling node is a variable, this method sets the Value attribute. The varianttype parameter is used to parse the built-in Python types to VariantTypes using the ua.VariantType enumeration. Optionally, the argument could be passed directly using the ua.Variant(value) constructor.

delete_reference((target, reftype, forward=True, bidirectional=True))

This method tries to delete a reference from the calling node. The ReferenceType must be specified by the reftype parameter, and the destination of the reference must be specified by the target parameter. The parameter reftype can be passed using the enum ua.ObjectIds and the target node must be passed by using a NodeId.

add_reference(target, reftype, forward=True, bidirectional=True)

This method adds a reference to the calling node and has the same arguments as its inverse method above.

3.2.3 uamethod(func)

This method decorator automatically converts arguments and the output of a function to and from variants. This conversion ensures that a python method linked to an OPC UA method will get arguments of built-in Python types, and the outputs will be converted to variants again using the ua.Variant(value) constructor as described above.

3.2.4 ua.StatusCode(code)

This constructor creates a status code object that can be returned to a client from an OPC UA method to report that an error has occurred. The code parameter should be passed using the ua.StatusCodes enumeration.

3.3 Creation of a new OPC UA namespace

The Siemens OPC UA Modelling Editor (SiOME) was used to create a new information model matching the problem statement. On top of the OPC UA Standard model, the OPC UA Devices model and the OPC UA for Robotics model, the fourth nodeset file Tuw.Auto.MitsubishiElectricRobot.NodeSet2.xml was defined representing the problem-specific information model. The model is defined to use the namespace index four because the namespace index one is reserved for dynamically created nodes of the server, as mentioned above. This tool is convenient because it operates only on XML files and saves only XML files without having any internal, proprietary format. XML is also the format the python-opcua framework uses for importing information models. SiOME also offers a pretty good search function to find nodes in currently open XML files. All mentioned, and other features can be found on [17]. The newly created information model instantiates the MotionDeviceSystem, an object of type MotionDeviceSystemType under the node DeviceSet using the HasComponent reference as stated in the OPC UA for Robotics specification [10, page 28]. Some types were added and modified to better match the real-world system. Also, some static nodes were added to some components of the instantiated MotionDeviceSystem like model type and manufacturer name of various parts. The additional functionality includes the ability to start/stop task controls, call safety functions, and getter/setter methods in the ParameterSet of the controller. The modifications were necessary because the OPC UA for Robotics specification is not complete and the first part that was released only focusses on monitoring, not on the operation. For the whole list of supported and implemented features, refer to the Functions of the implementation section.

3.4 Proprietary telnet commands of the robot controller

By far, the most importing document while implementing the OPC UA server was the manual [18]. It describes how to form the request and how the response will be formed. The exact structure can be looked up at [18, page 8]. The request is formed [<Robot No.>];[<Slot No.>];<Command><Argument>. The robot number has a range from zero to three, and the slot number can have values from zero to thirty-three. If unsure which robot number and slot number to pick, take one, for both. The command and argument section differs in each command and will be described below. The response is either formed QoK<Answer> or QeR<Error No.>; the first one signals success, and the second one signals that an error has occurred. The error number can be translated using the troubleshooting manual [19]. The answer in case of a successful response also differs in each command and will also be described below. The service described above can easily be accessed using a standard telnet client. Only the robot controller's IP and correct port must be provided. The following list will contain only clarifications about the argument part of the request and the answer part of a successful response. The command equals the headline. For example, calls, refer to the Appendix chapter. All the proprietary commands used for the implementation, including a short description of their functionalities and syntax, are:

3.4.1 CNTLON

This command has no arguments and no answer and enables the operation right for the connected client. This right is needed when executing the SRVON/SRVOFF, RUN and SLOTINIT commands. Otherwise, an error number will be returned in the error response. For more detailed information, refer to [18, page 17].

3.4.2 TIME

This command has no arguments and returns <DATE><TIME>. The date is formatted as yy-mm-dd and the time is formatted as hh:mm:ss. For more detailed information, refer to [18, page 64].

3.4.3 SLOTINIT

This command has no arguments and no answer. By executing this command, all program slots will be reset. Therefore, all the robot programs in the slots will resume execution from the start. This command must be called before a call to PRGUP and PRGLOAD=. For more detailed information, refer to [18, page 44].

3.4.4 PRGUP

This command has no arguments and no answer. The controller loads the next program in the list of available programs into the slot specified in the command. For more detailed information, refer to [18, page 37].

3.4.5 PRGRD

This command has no arguments and returns the name of the program loaded in the specified slot. For more detailed information, refer to [18, page 37].

3.4.6 OVRD

This command has no arguments and returns the operation override in percent. The override is a factor slowing down the programmed robot movement speed of a program. For more detailed information, refer to [18, page 41].

3.4.7 OVRD=

This command takes a single integer as argument and returns the set override value as the answer. The integer argument specifies the override and should have a value between one and one-hundred. For more detailed information, refer to [18, page 40].

3.4.8 PAR

This command takes a parameter name as argument and returns <Parameter name>;<Value>;<Value count> as answer. The Value field describes the current value of the parameter. For more detailed information, refer to [18, page 85].

3.4.9 PAW=

The argument of this command consists of <Parameter name>;<Value> and the argument itself has no answer. The parameter of the robot controller matching the Parameter name field will be changed according to the Value field. For more detailed information, refer to [18, page 86].

3.4.10 STOP

This command has no arguments and no answer. It will stop an eventual running program execution and therefore also the moving robot. For more detailed information, refer to [18, page 42].

3.4.11 CSTOP

This command has no arguments and no answer. It will stop the executing robot program at the next cycle start. For more detailed information, refer to [18, page 43].

3.4.12 STATE

This command has no arguments and returns <Program name>;<Line no.>;<Override>; <Edit sts.>;<Run sts.><Stop sts.><Error no.>; <Step no.>;<Mech info.>;;;;;;;;;<Task prg.name>;<Task mode>;<Task cond.>;<Task pri.>;<Mech no.> as the answer. The most important member in this answer is the integer run status specified as Run sts. in the manual. If this number is written in binary, the set bit at index six represents if a program is currently started. For more detailed information, refer to [18, page 52].

3.4.13 MOVSP

This command has no arguments and no answer. It makes the controller move to the position specified by the parameter JSAFE. For more detailed information, refer to [18, page 47].

3.4.14 PRGLOAD=

This command takes the program name as an argument and has no answer. The controller will load the program specified by the program name into the corresponding slot. For more detailed information, refer to [18, page 36].

3.4.15 RUN

This command takes the argument formed as <Program name>;<Mode> and has no answer. The program name can be omitted to execute the program under the selection. The Mode field is an integer that can have the values zero for a repeated start, therefore continuous operation, or one for a cycle start, which only executes the program until the beginning of the next cycle. For more detailed information, refer to [18, page 41].

3.4.16 RSTALRM

This command has no arguments and no answer. The current active controller error is reset when executing this command. For more detailed information, refer to [18, page 43].

3.4.17 RSTPRG

This command has no arguments and no answer. It resets the program loaded into the specified slot. For more detailed information, refer to [18, page 44].

3.4.18 SRVON

This command has no arguments and no answer. The servo power supply is turned on, and this enables joint movement. For more detailed information, refer to [18, page 40].

3.4.19 SRVOFF

This command has no arguments and no answer. The servo power supply is turned off, and this disables joint movement. For more detailed information, refer to [18, page 40].

3.4.20 ERRORLOG

This command takes the history number as argument and returns <DATE>;<TIME>; <Error no.>;<Error contents>;<Error level>;<Program name>;<Line no.>;<Error detail no.> as answer. The most important part of the answer is the Error contents field, which is an English error description. The argument history number could be specified to TOP for the newest error, END for the oldest error or +1/-1 to scroll backward/forward in time. For more detailed information, refer to [18, page 68].

3.5 Validation of the implementation

The UaExpert OPC UA Client from Unified Automation was used to verify the server implementation. This GUI tool offers an appropriate layout to discover the nodes, and all errors are printed to the log, which helps a lot in the debugging process. The software is free and can be downloaded on [20]. Additional not-mentioned features can be found there as well. One test run included read operations to all written variables, and all implemented methods were called in different states of the robot system. A simple robot program was written, that keeps the robot moving between positions, which are defined in the corresponding position list to ease testing. The MB5 and POS files make up the described robot program that can be found in the Appendix chapter. The ResumeProgram OPC UA method of the specific task control object has to be called with the parameter Repeated set to enabled to run the robot program continuously.

3.6 Solution to move the robot in real-time

The robot controller also supports a real-time external control function, which allows a computer to control a robot controller that is connected via a one-on-one cross cable. At first, the robot controller must open a communication line. Then the robot controller has to execute an MXT command somewhere in the robot program code. While executing the MXT command, the robot motion movement control can retrieve the position data from the personal computer in real-time every approximately seven milliseconds and move to the commanded position. This period is called the motion movement control cycle. The external control function can easily be set up by following the tutorial starting on [21, page 3-9]. The tutorial is somewhat time-consuming and therefore not explicitly described. The information about the real-time external control function is taken from [21, page 3-9,4-5]. Real-time communication provides guaranteed latencies. This predictable behavior is required if the motion sequence of the industrial robot is dependent on various other devices' movements used in an industrial process. The computer, which controls the industrial robot, has to compute the motion sequence for the next motion movement control cycle and must consider all the movements of all other industrial devices interleaving in this industrial process during the computation. If computations regarding motion sequences are made, computers need a sense of time and also an upper bound of signal delay to coordinate all the computed movements. Real-time communication is used to accomplish these requirements. Therefore, if the movement of the robot is dependent on many other devices, it is better to control the robot externally by using the real-time external control function. Complex temporal and spatial dependencies cannot be satisfied if the robot is controlled internally by its controller following a sequential robot program. The thesis' problem statement does not require the usage of the external robot control function, but for complex industrial processes, this function gives the programmer full control over the robot's motion sequence.

CHAPTER 4

Implementation

4.1 Implementation overview

The SysML block diagram in Figure 4.1 shows all hardware and software components of the implementation and makes the information flow between all components visible. The type of information flow is also presented in the figure. It also shows how OPC UA clients interact with the whole system. The mentioned OPC UA information models were imported in the opcuaserver.py script.

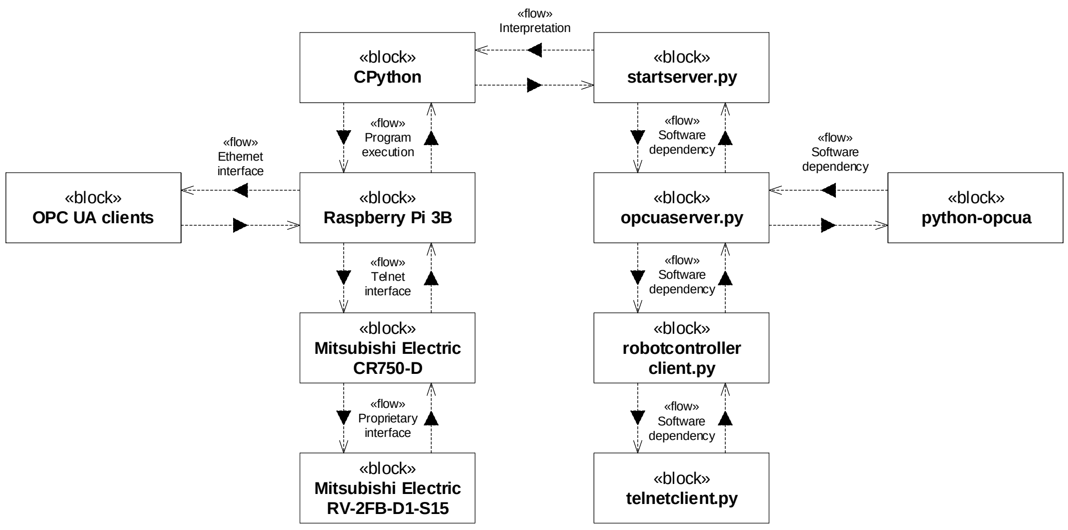

«flow» Interpretation
«block» CPython
«block» startserver.py
«flow» Program execution
«flow» Software dependency
«flow» Ethernet interface
«flow» Software dependency
«block» OPC UA clients
«block» Raspberry Pi 3B
«block» opcuaserver.py
«block» python-opcua
«flow» Telnet interface
«flow» Software dependency
«block» Mitsubishi Electric CR750-D
«block» robotcontroller client.py
«flow» Proprietary interface
«flow» Software dependency
«block» Mitsubishi Electric RV-2FB-D1-S15
«block» telnetclient.py

Figure 4.1: Implementation components and their interactions

4.2 Hardware

4.2.1 Raspberry Pi 3B

The Raspberry Pi 3B used was from revision 1.2 and was chosen to host the OPC UA server and to process the communication with the robot controller. This single-board computer offered 1 gigabyte of RAM and was manufactured by Sony UK. The 64-bit processor of this Raspberry Pi has the model name BCM2837 and was manufactured by Broadcom. The chip features the same underlying architecture as its predecessor BCM2836 but uses a quad-core ARM Cortex A53 (ARMv8) cluster instead of an ARMv7 quad-core cluster. The clock frequency measures 1.2 gigahertz, and the integrated VideoCore IV is clocked at 400 megahertz. The processor information was taken from [22]. As this Raspberry Pi model offers a 100Base-T Ethernet interface, the communication to the robot controller is accomplished via Ethernet, and the OPC UA server is also only accessible via Ethernet. The single-board computer runs the default Raspbian operating system. The hardware specification was looked up based on the revision code a02082 on the website [23].

4.2.2 Mitsubishi Electric CR750-D

As stated in [24, page 2-6] the installation of this robot controller consists of the controller itself, four proprietary connectors, ferrite cores for the emergency stop wiring, a noise filter, and different fuses. Two out of the four proprietary ports are used to exchange the external emergency stop signals, the door switch signals, the switching operation mode signals, emergency stop active signals, safety measure signals for teaching mode and the mode output signals which determine the controller operation mode. Another proprietary port serves for connecting a decoder when using the tracking function. The last proprietary port is used to exchange monitoring signals from the robot to be operated by our controller. Our installation also includes a teaching pendant (T/B) which is used to operate on joints directly to get the coordinates of specific points that are used in the position lists of the robot programs. The controller must be in manual mode (switching the operation mode switch on the front of the controller in the correct position), and the T/B enable switch must be pressed on the backside of the teaching pendant to use this device. The T/B is also used for setting internal parameters of the controller to alter the controller's behavior, for example, the speed override, which is a factor in slowing down the programmed operation speed. There is also the possibility to press the emergency stop button on the T/B to stop the controller operation. The thesis' OPC UA server implementation will cover some of the teaching pendant's functions. It should also be mentioned that the controller has, of course, an Ethernet port to be able to connect to the local network as described in [21, page 2-1].

4.2.3 Mitsubishi Electric RV-2FB-D1-S15

This robot has six degrees of freedom as described in the table on [25, page 2-9 to 2-10]. It operates via AC servo motors, has an absolute decoder, an absolute maximum velocity of 4.95 meters per second and can carry a maximum load of three kilograms. The robot needs a power cable and a signal cable which are both connected to the controller. The correct ports on the controller to connect the wires are CN1 (motor power) and CN2 (motor signal) as taken from [25, page 2-33].

4.3 Program structure of the implementation

The main project folder consists of the three folders model, src, and script. All four XML information models, which were described before, are in the model folder. These models are going to be imported by the OPC UA server framework. The four Python source files are in the src folder. The script folder contains two shell scripts responsible for the deployment. They automatically load the newest files from Google Drive and start the OPC UA server afterward. These scripts were convenient while operating with the Raspberry Pi because transferring files via the cloud is much easier than with command-line tools like scp. Figure 4.2 shows a sequence diagram representing the invocation of the OPC UA method GetParameter by an OPC UA client. This figure should help to understand the information flow of the implementation described in the following sections. The Python file startserver.py is not present in the diagram because it only starts the OPC UA server and has no further functionality. The framework python-opcua maps the requested method invocation from the OPC UA client to a Python method of the server class in opcuaserver.py and also builds the response for the OPC UA client by using the return value of the corresponding Python method.

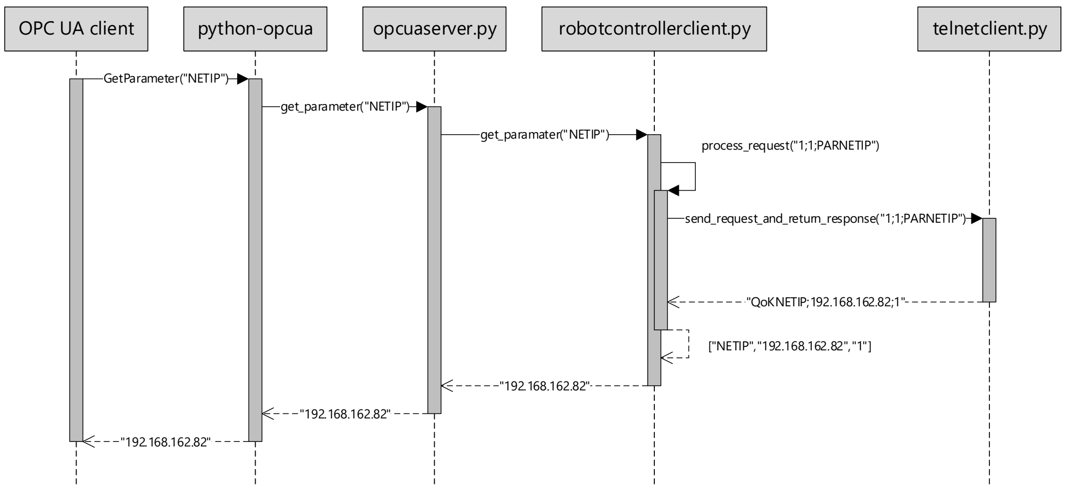


Figure 4.2: Sequence diagram representing a sample OPC UA method invocation

The following paragraphs will describe the program structure based on the Python source files:

4.3.1 telnetclient.py

In this Python file, the TelnetClient class is defined, which is used by the robot controller client class to communicate with the robot controller over TCP/IP. The Python module telnetlib was used, which already implements a standard telnet client. The constructor saves IP address, port, timeout, setup_method, test_command, and the encoding to instance variables. Furthermore, a lock is created to get mutual exclusion when accessing the telnet client. Then the telnet instance is created in the constructor using the method connect_to_server. This method creates a new telnet client by handing over the IP address, the port, and the timeout for the connection build-up. There is also a ConnectionMaintainer thread started in the constructor that checks the connection periodically using the test_command passed with the constructor. If the connection is broken, the connect_to_server method is called, and after that, the setup_method method is called too, which issues the first command to the telnet server. The robot controller client specifies this command. Then there is the send_request_and_return_response method, which clears all buffers using the read_very_eager method and then sends the request to the telnet server, the robot controller. Then the client waits for a specific timeout. This timeout can be specified via the standard timeout as a parameter of the constructor and an individual timeout factor. After the waiting period, the telnet client tries to read everything it can without blocking in I/O, and this string is returned as a response. Since there is no clear separator between two responses, the implementation uses a timeout. The request must be encoded, and the response must be decoded according to the given constructor parameter encoding.

4.3.2 robotcontrollerclient.py

In this Python file, the RobotControllerClient class is defined, which is responsible for sending the right strings to the robot controller to implement concrete application logic. The OPC UA server must only call the methods of the robot controller client to implement the abstract model in the address space. There are many class constants in the RobotControllerClient class like the TEST_COMMAND, the ENCODING, the START_SEQUENCE_SUCCESS, and the DELIMITER string. First, the constructor saves the IP address and the port as instance variables. Then, the TelnetClient class used for the communication with the robot controller is instantiated using the constructor described above. After this instantiation, the operation right is retrieved using the setup_controller method, which calls the CNTLON command. Then the list of available programs is initialized in the constructor, which is used by the OPC UA server to create task control nodes dynamically. First, the SLOTINIT command is issued to the controller using the initialize_all_slots method to make to call to the following commands possible and valid. Since there is no convenient way specified in the telnet manual [18] for getting a list of all programs stored on the controller, the list is scrolled through with the PRGUP command in a specified slot until the program at start-up is reached. The command PRGRD retrieves the program name currently loaded in the specified slot. Then, the constructor returns. Every time the controller issues a command to the robot controller, the command is sent via the process_request method as parameter req, which uses the send_request_and_return_response method from the telnet client. It is also possible to pass a timeout_factor in the calling method. After the response is returned by this method, it is checked if it starts with START_SEQUENCE_SUCCESS. If it does, the rest of the response is split every semicolon, and the resulting list is returned. Otherwise, an error must have occurred, and the application signals this by raising a RobotControllerError. This error takes a single value from the ua.StatusCodes enumeration as a parameter that will be sent to the OPC UA client by the OpcUaServerForRobotController class, which is described below. The value ua.StatusCodes.BadCommunicationError is used as a status parameter to the RobotControllerError in this case because most of the time, a broken connection will cause the exception to be raised. Optionally, also an error code can be passed to the RobotControllerError, which is done in this case by using the rest of the response string when cutting away the START_SEQUENCE_ERROR. This result signals the error code as described in the above chapters. Most of the methods of the RobotControllerClient class are one-liners that return some part of the response or do not return anything when only the state of the robot controller is changed. They only map the function calls to the robot controller methods in a one-to-one fashion. All the methods that are sending requests to target robot number one and slot number one if the corresponding parameters are omitted, and therefore, the default parameters are used. The turn_servos_on method includes a delay because the enabling process takes some time. The turn_servos_off method specifies a timeout_factor because the robot controller needs more time to answer than is specified in the standard timeout. The only more sophisticated method of the RobotControllerClass is the resume_program method. This method first checks if the given program_name as a parameter is valid, therefore if the parameter is in the program_list that was filled before. If not a RobotControllerError is raised with the status code ua.StatusCodes.BadInvalidArgument. Also, if the robot controller is currently executing a robot program, no more robot programs can be started, therefore the ua.StatusCodes.BadInvalidState code is passed on to the error as a parameter. If a program_name is specified that is different from the program name currently loaded in the slot; all slots must be initialized, and the correct program must be loaded into the correct slot. Then the servos are turned on, and then the program is started using the above-described RUN command with correct parameters.

4.3.3 opcuaserver.py

In this Python file, the OpcUaServerForRobotController class is defined. The first thing in the constructor of the OpcUaServerForRobotController class is the instantiation of the RobotControllerClient class to be able to call its methods, which are described above. The IP address and the port of the robot controller are passed on from constructor to constructor. The next step is the instantiation of the Server class from the python-opcua framework. Then, the endpoint and the name of the Server instance are set. As the specified endpoint starts with "opc.tcp://", the OPC UA Binary TCP transport is used. Next, the security policy is set to no encryption. The next step is to import all four information models described in the above chapters and integrate them into the server's address space. As the framework imports the OPC UA standard model itself, some will ask why this file is imported twice. This second import is necessary because the imported standard model is in version 1.04 and from the publication date 2017/11/22, which does not contain all the nodes required by the OPC UA Devices and OPC UA Robotics nodeset files. Therefore, a minimal necessary nodeset amendment file was created, which defines all the nodes that the two other nodeset files needed. The nodes were looked up in version 1.04 and publication date 2019/05/01 of the OPC UA standard model. It was looked up on how to change the imported standard nodeset file, but this is rather complicated, and most likely the version of the nodeset file is going to be updated in a future release. After that, all Python methods that are not concerning the dynamically created task controls of the robot controller can be linked statically to the OPC UA methods. This linking is done using the link_method method of the OpcUaServerForRobotController class, which only calls the link_method method from the framework with the correct parameters. The task control nodes are then created dynamically by executing the generate_task_controls method. At first, a placeholder node from the OPC UA for Robotics nodeset file is deleted. Then for every program loaded into the robot controller, a single task control node is created using the add_object method, which instantiates the TaskControlType of the model. The calling node is the TaskControls node of the ControllerType instance. As the framework uses an Organizes reference as default to link the added object to the parent, the ReferenceType was changed to HasComponent to be conformant to the OPC UA for Robotics companion specification. Then the Python methods reset_program, stop_at_next_cycle and resume_program are linked to newly created nodes. Then the optional motion device node is deleted from the newly created task control object. The name of the task control is also set accordingly. This part of the script makes excessive use of the get_child method of the NodeClass to navigate through the address space. When calling the start method, not only the Server instance from the framework is started, but also the PeriodicWorker thread, which executes the periodic_routine method periodically. This call actualizes the OPC UA variables IsMoving and SpeedOverride of the controller in the address space. Also, this call updates the TaskProgramLoaded variable of every task control. All the OPC UA methods use the uamethod decorator of the framework and delegate the requests to the RobotControllerClient instance. Every Python method that was linked to an OPC UA method excepts the RobotControllerError and send a corresponding error response to the OPC UA client using the error_response method. The three methods linked to the task controls, check if the calling parent node models the current running program, otherwise an error response is returned with the status_code ua.StatusCodes.BadInvalidState.

4.3.4 startserver.py

In this Python file, the OpcUaServerForRobotController class is instantiated and started in the last line. The sys module is imported to be able to read the command line arguments. The IPy module is used to check if the given IP addresses are valid. The third import is the OpcUaServerForRobotController class. Then, the two constants MAX_PORT and MIN_PORT are defined, which are used in error message strings but also in the check_port_number function. After the definition of this function, the length of the argument list is checked. If we have not got all five command line parameters, the program exits with the USAGE string. Only if the correct number of arguments is passed, all arguments are checked for correct values. If an argument, which is passed from the command line, cannot be converted to a port or an IP address, the program exits with a corresponding error message. If all given parameters are valid, they are passed on to the constructor of the OpcUaServerForRobotController class to instantiate it. After the constructor returns, the start method of this instance is called to make the server visible to OPC UA clients on the same network.

4.4 Functions of the implementation

All functions that are mentioned in the current version of the OPC UA for Robotics companion specification but are not discussed in this section will not work as expected. From the Root node, navigate to the Objects Node, then to the DeviceSet node and then finally to the MotionDeviceSystem node following the Organizes references. The MotionDeviceSystem node represents our robot system. This node has three folders as components: the Controllers folder, the MotionDevices folder, and the SafetyStates folder. All folders only contain one object, which is a single Controller, a single MotionDevice, and a single SafetyState object. The functionality of the implementation will be discussed object by object. The following figures were created using the standard OPC UA graphical notation. Table 4.1 describes the modeling of nodes and Table 4.2 describes the modeling of references. The OPC UA figures only contain parts of all the nodes and references of the address space. Figure 4.3 shows the ObjectType hierarchy of the implementation, and the information model they belong to, Figure 4.4 shows the VariableType hierarchy, which only contains nodes of the OPC UA Standard information model. Figure 4.5 shows an overview of the address space, which was described above.

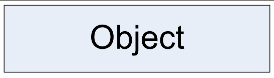
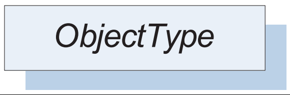
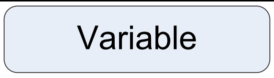
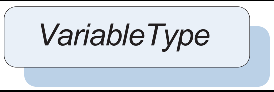
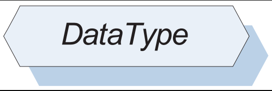
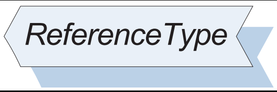
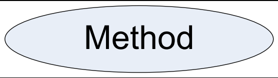
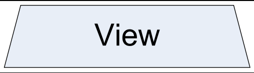
NodeClass	Graphical Representation	Comment
Object		Can contain the TypeDefinition separated by "::", e.g., "Object1::Type1"
ObjectType		Abstract types use italic, concrete types not
Variable		Can contain the TypeDefinition separated by "::", e.g., "Variable1::Type1"
VariableType		Abstract types use italic, concrete types not
DataType		Abstract types use italic, concrete types not
ReferenceType		Abstract types use italic, concrete types not
Method		–
View		–

Table 4.1: OPC UA graphical notation of nodes based on NodeClasses [5, page 328]

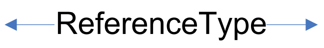
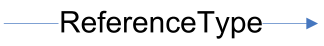
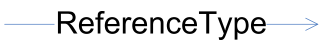
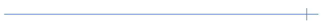
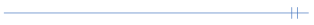
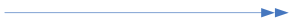
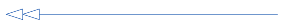
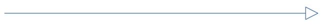
ReferenceType	Graphical Representation
Any symmetric ReferenceType	
Any asymmetric ReferenceType	
Any hierarchical ReferenceType	
HasComponent	
HasProperty	
HasTypeDefinition	
HasSubtype	
HasEventSource	

Table 4.2: OPC UA graphical notation of references based on ReferenceTypes [5, page 329]

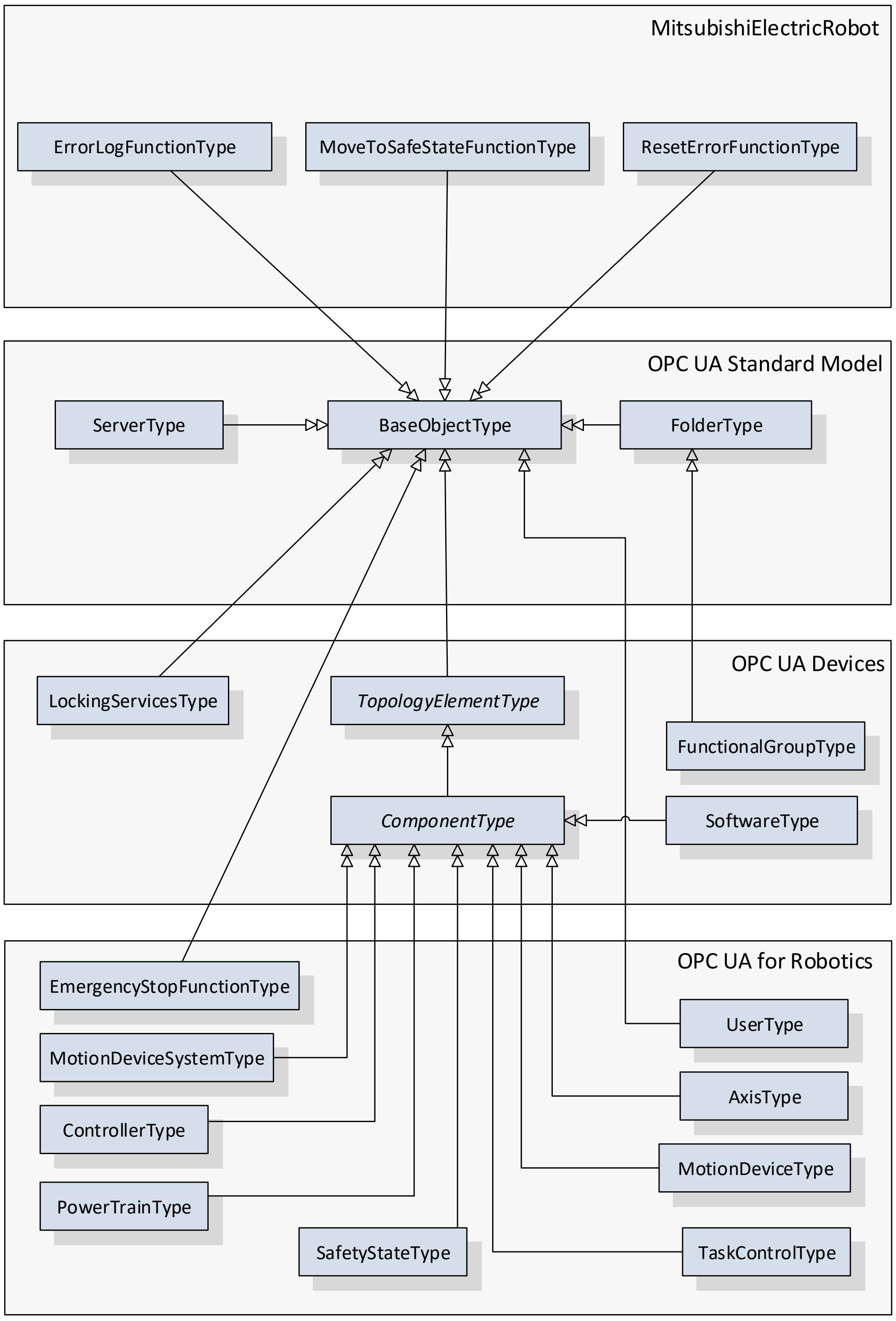


Figure 4.3: The ObjectType hierarchy of the implementation

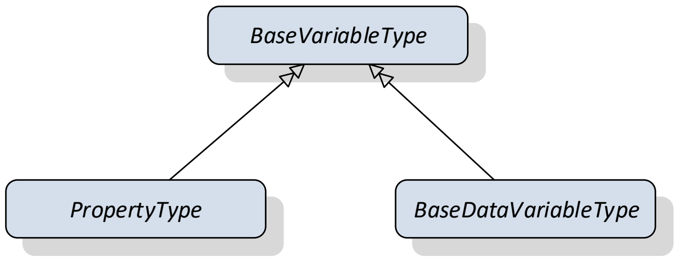


Figure 4.4: The VariableType hierarchy of the implementation

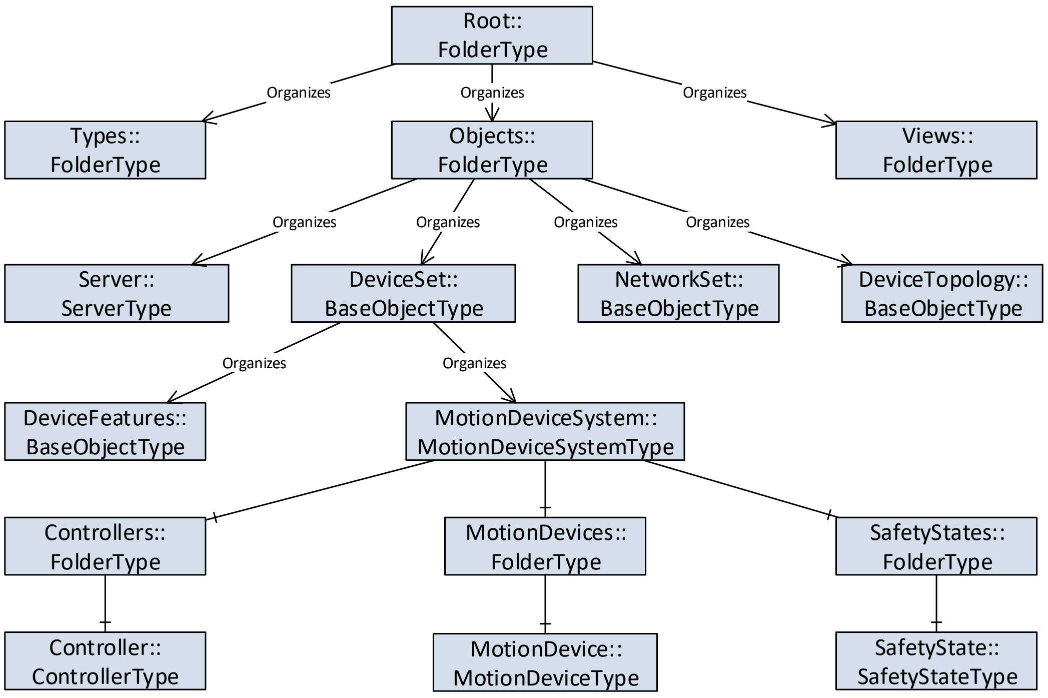

Root:: FolderType
Organizes
Types:: FolderType
Objects:: FolderType
Views:: FolderType
Server:: ServerType
DeviceSet:: BaseObjectType
NetworkSet:: BaseObjectType
DeviceTopology:: BaseObjectType
DeviceFeatures:: BaseObjectType
MotionDeviceSystem:: MotionDeviceSystemType
Controllers:: FolderType
MotionDevices:: FolderType
SafetyStates:: FolderType
Controller:: ControllerType
MotionDevice:: MotionDeviceType
SafetyState:: SafetyStateType

Figure 4.5: The complete address space of the OPC UA server

4.4.1 Controller

The Controller object offers a valid Model and Manufacturer variable. It also has a component called ParameterSet, which has itself three methods as components. The three methods are GetParameter, SetParameter, and SetOverride. GetParameter needs a parameter name to retrieve its value, SetParameter needs a parameter name and a value to set a value, and SetOverride only needs a value to set the override. The functionality should be clear as the methods use telling names. The controller itself also has the TaskControls folder as a component, which itself organizes the dynamically created TaskControls. Each TaskControl has a ParameterSet as a component, which has the two variables, TaskProgramLoaded and TaskProgramName. These two variables are filled by the application. TaskProgramLoaded is true if the program is currently loaded into the slot. Each TaskControl also supports the methods ResetProgram, ResumeProgram, and StopAtNextCycle, which are components of the MethodSet object. The MethodSet is a component of the task control object itself. A paused program can be reset to the start by executing the ResetProgram method. The ResumeProgram method can be used with the parameter Repeated, which should be true for continuous operation, to start or resume a program. If the method StopAtNextCycle is called, the program in execution finishes the current cycle and then stops the program. The controller object part of the address space and a task control object are shown in Figures 4.6 and 4.7.

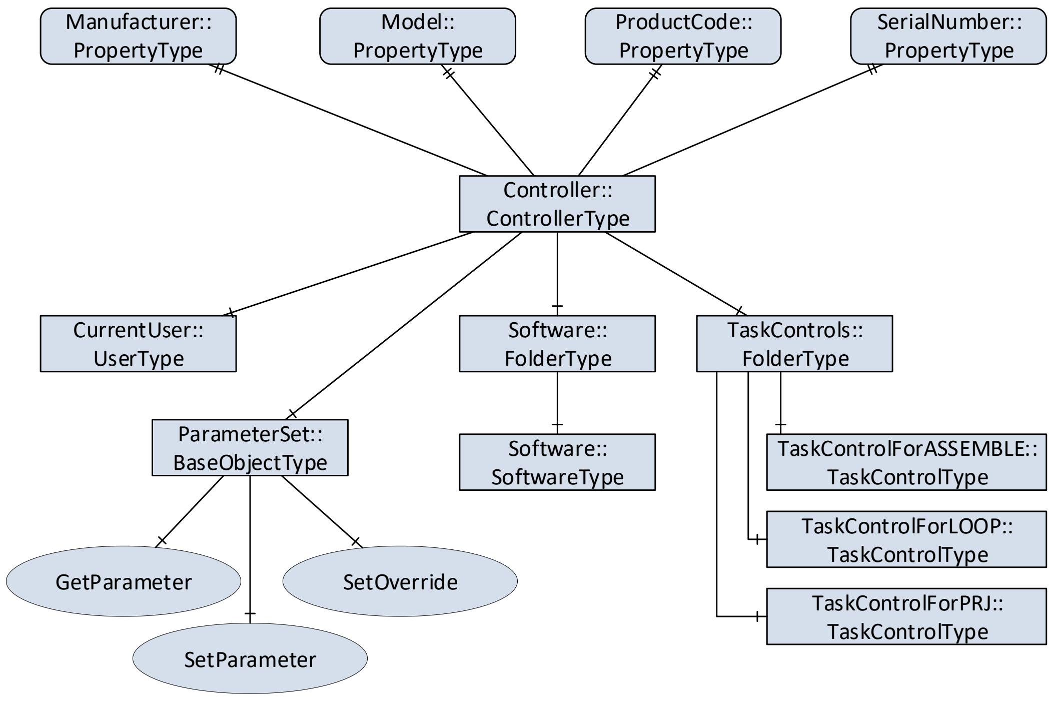

Manufacturer:: PropertyType
Model:: PropertyType
ProductCode:: PropertyType
SerialNumber:: PropertyType
Controller:: ControllerType
CurrentUser:: UserType
Software:: FolderType
TaskControls:: FolderType
ParameterSet:: BaseObjectType
Software:: SoftwareType
TaskControlForASSEMBLE:: TaskControlType
TaskControlForLOOP:: TaskControlType
TaskControlForPRJ:: TaskControlType
GetParameter
SetOverride
SetParameter

Figure 4.6: The controller object part of the address space

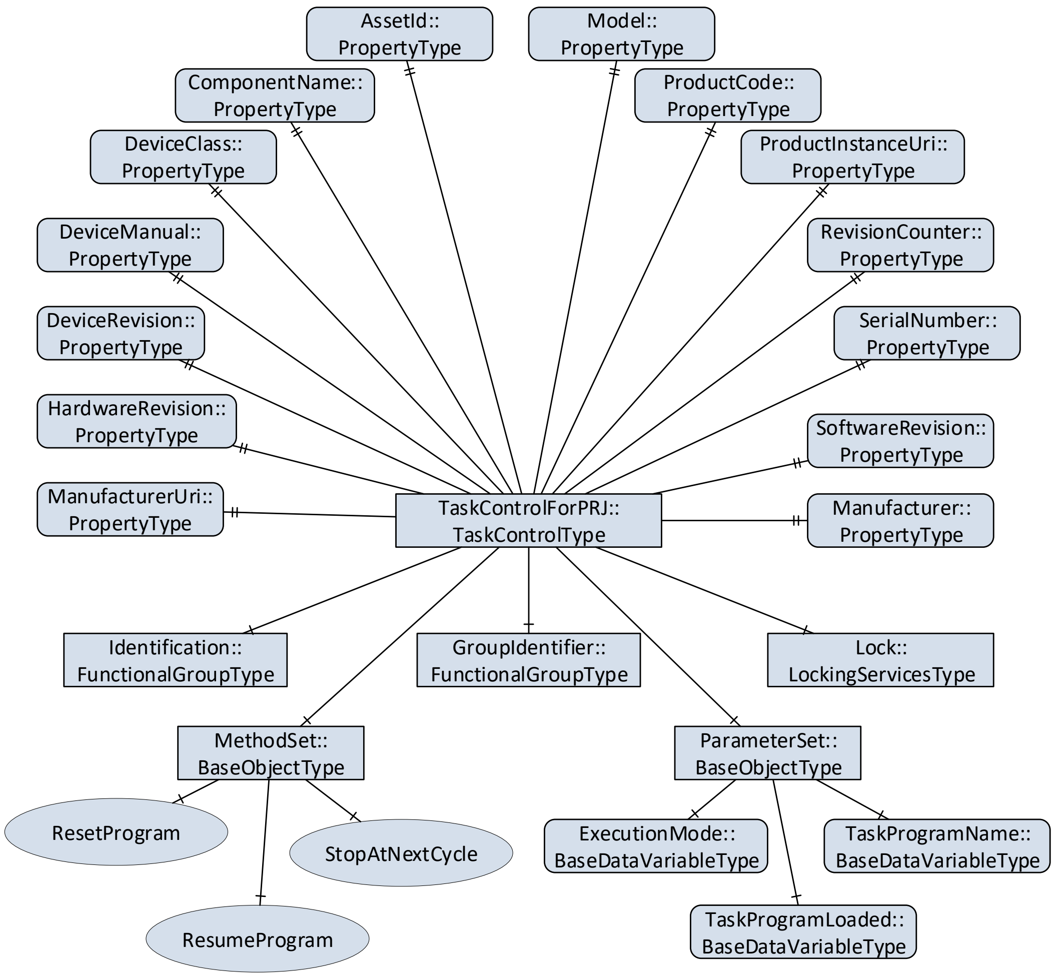

AssetId:: PropertyType
Model:: PropertyType
ComponentName:: PropertyType
ProductCode:: PropertyType
DeviceClass:: PropertyType
ProductInstanceUri:: PropertyType
DeviceManual:: PropertyType
RevisionCounter:: PropertyType
DeviceRevision:: PropertyType
SerialNumber:: PropertyType
HardwareRevision:: PropertyType
SoftwareRevision:: PropertyType
ManufacturerUri:: PropertyType
TaskControlForPRJ:: TaskControlType
Manufacturer:: PropertyType
Identification:: FunctionalGroupType
GroupIdentifier:: FunctionalGroupType
Lock:: LockingServicesType
MethodSet:: BaseObjectType
ParameterSet:: BaseObjectType
ResetProgram
StopAtNextCycle
ResumeProgram
ExecutionMode:: BaseDataVariableType
TaskProgramName:: BaseDataVariableType
TaskProgramLoaded:: BaseDataVariableType

Figure 4.7: The task control object part of the address space

4.4.2 MotionDevice

The MotionDevice object offers a valid Model and Manufacturer variable. It also has a component called ParameterSet, which itself organizes two variables. The SpeedOverride variable holds the current override value, and the IsMoving variable holds a Boolean value that indicates if the MotionDevice moves or not. The motion device object part of the address space is shown in Figure 4.8.

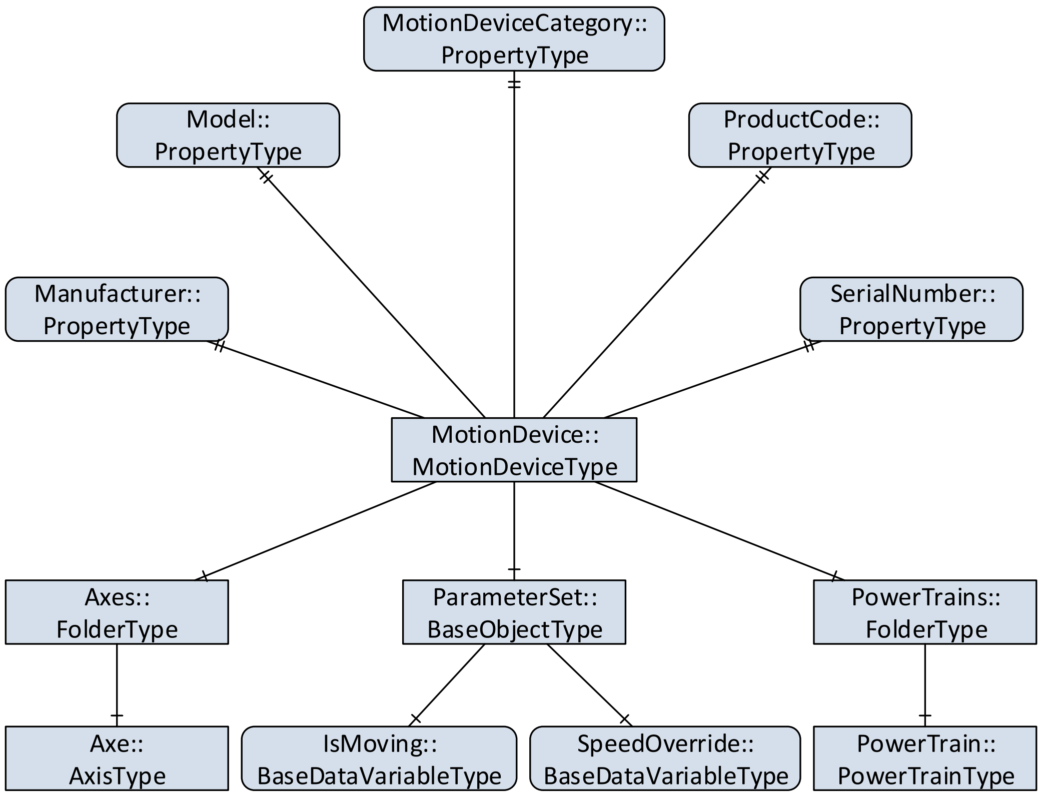

MotionDeviceCategory:: PropertyType
Model:: PropertyType
ProductCode:: PropertyType
Manufacturer:: PropertyType
SerialNumber:: PropertyType
MotionDevice:: MotionDeviceType
Axes:: FolderType
ParameterSet:: BaseObjectType
PowerTrains:: FolderType
Axe:: AxisType
IsMoving:: BaseDataVariableType
SpeedOverride:: BaseDataVariableType
PowerTrain:: PowerTrainType

Figure 4.8: The motion device object part of the address space

4.4.3 SafetyState

The SafetyState object offers four folders, which contain objects that abstract safety functions. The four folders are the EmergencyStop Functions, the ErrorLog Functions, the MoveToSafeState Functions and the ResetError Functions folder. The structure of all the objects in the folders is the same. All the objects have a method as a component called CallFunction. All four methods have no input parameters. The functionality of the methods is the same as the name of the objects in the folder. There is the StopImmediately object with a function to stop robot movement and a GetMostRecentError object with a function to retrieve the last critical error as English text. There is also a MoveToSafePosition object with a function to make the robot move to the position specified by the robot controller parameter JSAFE and the ResetError object with a function to reset a currently active error state. The safety state object part of the address space is shown in Figure 4.9.

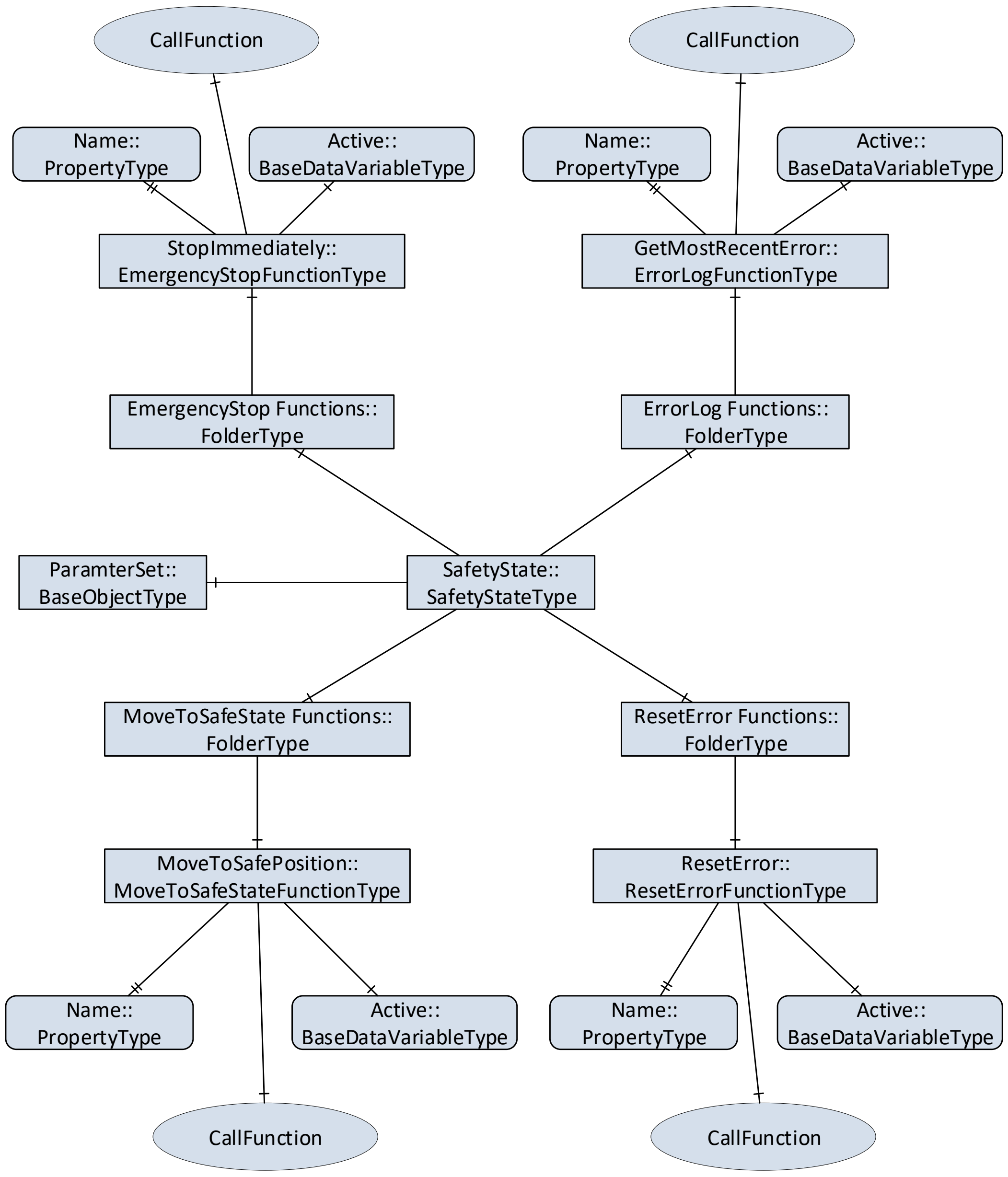


Figure 4.9: The safety state object part of the address space

CHAPTER 5

Critical reflection

5.1 Comparison with related work

This thesis was compared to the thesis [26], which implements an OPC UA server for a numerical control machine. The author had to define a custom information model because there was no companion specification available for this industry domain at this time. The model is described beginning on [26, page 18]. The ability to use the OPC UA for Robotics information model guided the implementation process of this thesis because it provided a basic structure. Also contrary to this thesis, the OPC UA Server for the numerical control machine was constructed with a closed source server framework from Unified Automation GmbH. This was mentioned in [26, page 18]. Also mentioned on the same site of the comparing thesis, the used programming language was C++. The other thesis took the conservative approach by using a low-level programming language and a closed source framework. At this time, there were not as many usable open-source OPC UA server frameworks, so it was an understandable decision. Furthermore, the framework used by the other thesis supports code generation from the information model, which dramatically speeds up the implementation process and helps to establish a clean project structure.

5.2 Discussion of open issues

One of the open issues is the fact that a self-written amendment information model had to be imported to get all the nodes of the OPC UA standard model with the newer publication date. The used companion specification models need these additional nodes. A proper solution would be to update the imported OPC UA standard model nodeset file in the framework. This change could be committed, pushed, and merged to the python-opcua master branch via a pull request and will then be included in future releases. The Python package python-opcua does not provide any information in the essential

OPC UA Server object, which is a big downside of this framework. Furthermore, the robot programs read from the robot controller and used to instantiate the task controls are only determined at the start-up of the OPC UA server. Currently, there is no periodic update routine for the task controls. When executing the MoveToSafePosition OPC UA method, the servos are turned on, but upon completion, they are not turned off again. This issue also appears after other method calls. The servos stay active. The missing separator in the response of the robot controller is also an issue to the application. Currently, the end of a command is determined via a timeout, but this is a bad solution because it wastes resources. If stricter spatial and temporal requirements regarding the thesis' problem statement arise, it will be a future issue that the OPC UA server uses the proprietary telnet interface of the robot controller to control the robot's motion sequence. The supported robot program commands and the internal control by the robot controller will most likely not be able to meet these requirements. Therefore, the application should be updated to interface with the real-time external control function in the future to meet all future requirements.

CHAPTER 6

Summary and future work

6.1 Summary

This thesis shows, how to integrate a non-compliant device into the OPC UA architecture by adding a network gateway device with an OPC UA server implementation between the proprietary device interface and the OPC UA client. In the case of the thesis' problem statement, the target device is an industrial robot system. An OPC UA client connected to the same network as the gateway device can call services of the OPC UA server hosted by the gateway device, which was a Raspberry Pi in this implementation. The client has to use the supported transport method OPC UA Binary TCP. If a client issues a call to the server, the gateway device maps the OPC UA service to a suitable command understood by the proprietary device interface. In the case of the industrial robot system, this interface is a telnet server. This telnet server only supports a text-based, proprietary protocol. The information model of the implementation is based on the nodeset file of the OPC UA for Robotics companion specification. Not all nodes are working as described in the specification, and some nodes were added to solve the problem statement. The thesis' Python program, which adds functionality to the information model, is divided into four files. There is a startup script, a telnet client class to abstract the communication to the industrial robot system, a robot controller client class, which builds the proprietary telnet commands and a server class, which offers the OPC UA services described in the information model using the two previous classes. The open-source Python framework python-opcua is used in the server class to abstract the sophisticated software architecture of an OPC UA server.

6.2 Future Work

As new parts of the OPC UA for Robotics companion specification will be released in the future, the information model could be updated based on a new release of the companion specification and implemented with updated source files. Another task would be to create an implementation, which is fully compliant with the current version of the OPC UA for Robotics companion specification. Maybe some other open-source OPC UA server frameworks arise, which can be evaluated and used for the implementation of an updated OPC UA server application for this thesis's problem statement. There are also many open issues at the Github repository of the python-opcua framework, which are waiting for a pull request that fixes them. Notably, the framework's imported OPC UA standard model should be updated. Also, the OPC UA Server object should be filled with default values and should list the Standard Profile in the ServerCapabilities as default. There is also some future work to implement the periodic update routine for the task controls. As described in the open issues discussion section, the servos are sometimes not turned off properly. This issue should be fixed in the future. Furthermore, a better way has to be found to separate different robot controller responses. The current solution with a timeout is slowing down the whole application. Another future work would be to integrate the industrial robot system with the OPC UA server into a small scale production process in our lab to evaluate the usage of OPC UA in the context of an industrial automation plant. The controlling and monitoring of the industrial robot system could be done via OPC UA, so only a corresponding OPC UA client has to be implemented.

CHAPTER 7

Appendix

7.1 Bash scripts

```
#!/bin/bash
#delete old start server script
rm -rf /opc_ua_server_for_robot_controller/script/
    start_opc_ua_server_for_robot_controller.sh
#test name resolution
host "drive.google.com"
#download new start server script
wget "https://drive.google.com/uc?export=download&id=1
    Dnm2mi5h0FWqCHeWzO029lpN7XisxSdq" -O /
    opc_ua_server_for_robot_controller/script/
    start_opc_ua_server_for_robot_controller.sh
#change permissions of new start server script
chmod 777 /opc_ua_server_for_robot_controller/script/
    start_opc_ua_server_for_robot_controller.sh
#execute new start server script
/opc_ua_server_for_robot_controller/script/
    start_opc_ua_server_for_robot_controller.sh
```

Listing 7.1: download_script_and_start_server.sh

```
#!/bin/bash
#remove the source folder and create a new directory
rm -rf /opc_ua_server_for_robot_controller/src
mkdir /opc_ua_server_for_robot_controller/src
#load the source files into the source folder
```

```
wget "https://drive.google.com/uc?export=download&id=1
   VKtkIH0WAxGEpYv6w_t-3WdtkpiSmp3f" -O /
   opc_ua_server_for_robot_controller/src/opcuaserver.py
wget "https://drive.google.com/uc?export=download&id=1
   urfzHXuKsbhjV1yYcANURMQYAKfkwP5x" -O /
   opc_ua_server_for_robot_controller/src/robotcontrollerclient
   .py
wget "https://drive.google.com/uc?export=download&id=1
   iFTGB7agEejhrK3VUtrLUXkGqXC-iUUu" -O /
   opc_ua_server_for_robot_controller/src/telnetclient.py
wget "https://drive.google.com/uc?export=download&id=1
   zYGmbjHZTlKPuOGWs86gCYleAGrGaxzG" -O /
   opc_ua_server_for_robot_controller/src/startserver.py
#remove the model folder and create a new directory
rm -rf /opc_ua_server_for_robot_controller/model
mkdir /opc_ua_server_for_robot_controller/model
#load the information models into the model folder
wget "https://drive.google.com/uc?export=download&id=1
   dV9v0t1w5d9BtKorEbXY6je5qQlKm8DJ" -O /
   opc_ua_server_for_robot_controller/model/Opc.Ua.NodeSet2.xml
wget "https://drive.google.com/uc?export=download&id=1
   DtPKP6gu8aNQuxX28es_Gn-HZmPoOtpp" -O /
   opc_ua_server_for_robot_controller/model/Opc.Ua.Di.NodeSet2.
   xml
wget "https://drive.google.com/uc?export=download&id=11
   f845aFJnfpB2WPz2YgTgPFE_elcMcp5" -O /
   opc_ua_server_for_robot_controller/model/Opc.Ua.Robotics.
   NodeSet2.xml
wget "https://drive.google.com/uc?export=download&id=19
   tJz5HMbypFlZITh94XsWotozbXtZKyM" -O /
   opc_ua_server_for_robot_controller/model/Tuw.Auto.
   MitsubishiElectricRobot.NodeSet2.xml
#kill process that eventually uses port 4840
fuser -k 4840/tcp
#start opc ua server for robot controller
/usr/bin/python3.7 /opc_ua_server_for_robot_controller/src/
   startserver.py 192.168.162.84 4840 192.168.162.82 10003
```

Listing 7.2: start_opc_ua_server_for_robot_controller.sh

7.2 Python sources

```
import time
from threading import Thread
from opcua import Server, ua, uamethod

from robotcontrollerclient import RobotControllerClient as
    RCClient, RobotControllerError


class OpcUaServerForRobotController:

    def __init__(self, ip_os, port_os, ip_rc, port_rc):
        self.rc_client = RCClient(ip_rc, port_rc)

        self.opc_ua_server = Server()
        self.opc_ua_server.set_endpoint(f'opc.tcp://{ip_os}:{
            port_os}/')
        self.opc_ua_server.set_security_policy([ua.
            SecurityPolicyType.NoSecurity])
        self.opc_ua_server.set_server_name(self.__class__.
            __name__)
        self.opc_ua_server.import_xml('/
            opc_ua_server_for_robot_controller/model/Opc.Ua.
            NodeSet2.xml')
        self.opc_ua_server.import_xml('/
            opc_ua_server_for_robot_controller/model/Opc.Ua.Di.
            NodeSet2.xml')
        self.opc_ua_server.import_xml('/
            opc_ua_server_for_robot_controller/model/Opc.Ua.
            Robotics.NodeSet2.xml')
        self.opc_ua_server.import_xml(
              '/opc_ua_server_for_robot_controller/model/Tuw.Auto
                  .MitsubishiElectricRobot.NodeSet2.xml')

        self.link_method('ns=4;i=1115', self.
            move_to_safe_position)
        self.link_method('ns=4;i=1069', self.set_override)
        self.link_method('ns=4;i=1075', self.set_parameter)
        self.link_method('ns=4;i=1072', self.get_parameter)
        self.link_method('ns=4;i=1105', self.
            get_most_recent_error)
        self.link_method('ns=4;i=1119', self.stop_immediately)
        self.link_method('ns=4;i=1108', self.reset_error)

        self.generate_task_controls()
```

```
def link_method(self, nodeid, method):
    self.opc_ua_server.link_method(self.opc_ua_server.
       get_node(nodeid), method)

def generate_task_controls(self):
    self.opc_ua_server.get_node('ns=4;i=1038').delete(
       recursive=True)
    task_controls_object = self.opc_ua_server.get_node('ns
       =4;i=1022')
    for index, program_name in enumerate(self.rc_client.
       program_list):
        task_control_object = task_controls_object \
             .add_object(f'ns=1;i={index␣+␣1}', f'
                TaskControlFor{program_name}', 'ns=3;i=1011'
                )
        task_controls_object.delete_reference(
           task_control_object.nodeid, reftype=ua.ObjectIds
           .Organizes)
        task_controls_object.add_reference(
           task_control_object.nodeid, reftype=ua.ObjectIds
           .HasComponent)
        task_control_object.get_child(['3:<
           MotionDeviceIdentifier>']).delete(recursive=True
           )
        task_control_object.get_child(['2:ParameterSet', '
           3:TaskProgramName']).set_data_value(program_name
           )
        self.link_method(task_control_object.get_child(['2:
           MethodSet', '4:ResetProgram']).
                             nodeid, self.reset_program)
        self.link_method(task_control_object.get_child(['2:
           MethodSet', '4:StopAtNextCycle']).
                             nodeid, self.stop_at_next_cycle)
        self.link_method(task_control_object.get_child(['2:
           MethodSet', '4:ResumeProgram']).
                             nodeid, self.resume_program)

def periodic_routine(self):
    self.opc_ua_server.get_node('ns=4;i=1033').
       set_data_value(self.rc_client.get_override())
    self.opc_ua_server.get_node('ns=4;i=1122').
       set_data_value(self.rc_client.is_running())
```

```
        for index, program_name in enumerate(self.rc_client.
            program_list):
            self.opc_ua_server.get_node('ns=4;i=1022') \
                .get_child([f'0:TaskControlFor{program_name}',
                    '2:ParameterSet', '3:TaskProgramLoaded']) \
                .set_data_value(program_name == self.rc_client.
                    get_current_program())

    def get_program_name_from_parent_node(self, parent):
        return self.opc_ua_server.get_node(parent).get_parent()
            \
            .get_child(['2:ParameterSet', '3:TaskProgramName'])
                .get_data_value().Value.Value

    def start(self):
        self.opc_ua_server.start()
        PeriodicWorker(self).start()

    @staticmethod
    def error_response(status_code):
        return ua.StatusCode(status_code)

    @uamethod
    def set_override(self, parent, percentage):
        try:
            self.rc_client.set_override(percentage)
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)

    @uamethod
    def get_parameter(self, parent, name):
        try:
            return self.rc_client.get_parameter(name)
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)

    @uamethod
    def set_parameter(self, parent, name, value):
        try:
            return self.rc_client.set_parameter(name, value)
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)
    @uamethod
    def stop_immediately(self, parent):
        try:
            self.rc_client.stop_immediately()
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)

    @uamethod
    def stop_at_next_cycle(self, parent):
        if self.get_program_name_from_parent_node(parent) ==
            self.rc_client.get_current_program():
            try:
                return self.rc_client.stop_at_next_cycle()
            except RobotControllerError as rce:
                return self.error_response(rce.status_code)
        else:
            return self.error_response(ua.StatusCodes.
                BadInvalidState)

    @uamethod
    def move_to_safe_position(self, parent):
        try:
            return self.rc_client.move_to_safe_position()
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)

    @uamethod
    def resume_program(self, parent, repeated):
        try:
            program_name = self.
                get_program_name_from_parent_node(parent)
            return self.rc_client.resume_program(program_name,
                repeated)
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)

    @uamethod
    def reset_error(self, parent):
        try:
            return self.rc_client.reset_error()
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)
    @uamethod
    def reset_program(self, parent):
        if self.get_program_name_from_parent_node(parent) ==
           self.rc_client.get_current_program():
            try:
                self.rc_client.reset_program()
            except RobotControllerError as rce:
                return self.error_response(rce.status_code)
        else:
            return self.error_response(ua.StatusCodes.
               BadInvalidState)

    @uamethod
    def get_most_recent_error(self, parent):
        try:
            return self.rc_client.get_most_recent_error()
        except RobotControllerError as rce:
            return self.error_response(rce.status_code)


class PeriodicWorker(Thread):
    def __init__(self, server):
        Thread.__init__(self)
        self.server = server

    def run(self):
        while True:
            try:
                self.server.periodic_routine()
            except RobotControllerError:
                pass
            time.sleep(10)
```

Listing 7.3: opcuaserver.py

```
import time

from opcua import ua

from telnetclient import TelnetClient


class RobotControllerClient:
    TEST_COMMAND = '1;1;TIME'
    ENCODING = 'ascii'
    START_SEQUENCE_SUCCESS = 'QoK'
    START_SEQUENCE_ERROR = 'QeR'
    DELIMITER = ';'
    TIMEOUT = 0.1

    def __init__(self, ip, port):
        self.ip = ip
        self.port = port
        self.client = TelnetClient(self.ip, self.port, timeout=
            self.TIMEOUT, setup_method=self.setup_controller,
                                        test_command=self.
                                            TEST_COMMAND, encoding=
                                            self.ENCODING)
        self.setup_controller()
        self.program_list = []
        self.initialize_program_list()

    def initialize_program_list(self, robot_number=1,
        slot_number=1):
        while True:
            try:
                self.program_list.clear()
                self.initialize_all_slots(robot_number,
                    slot_number)
                while self.get_current_program(robot_number,
                    slot_number) not in self.program_list:
                    self.program_list.append(self.
                        get_current_program(robot_number,
                        slot_number))
                    self.scroll_up_to_next_program(robot_number
                        , slot_number)
                break
            except RobotControllerError:
                time.sleep(1)

    def initialize_all_slots(self, robot_number=1, slot_number
        =1):
        if not self.is_running(robot_number, slot_number):
            self.process_request(f'{robot_number};{slot_number
                };SLOTINIT')
        else:
            raise RobotControllerError(ua.StatusCodes.
            BadInvalidState)

def process_request(self, req, timeout_factor=1):
    res = self.client.send_request_and_return_response(req,
        timeout_factor)
    if res.startswith(self.START_SEQUENCE_SUCCESS):
        return res[len(self.START_SEQUENCE_SUCCESS):].split
            (self.DELIMITER)
    else:
        raise RobotControllerError(ua.StatusCodes.
            BadCommunicationError, res[len(self.
            START_SEQUENCE_ERROR):])

def scroll_up_to_next_program(self, robot_number=1,
    slot_number=1):
    self.process_request(f'{robot_number};{slot_number};
        PRGUP')

def get_current_program(self, robot_number=1, slot_number
    =1):
    return self.process_request(f'{robot_number};{
        slot_number};PRGRD')[0][:-4]

def get_override(self, robot_number=1, slot_number=1):
    return int(self.process_request(f'{robot_number};{
        slot_number};OVRD')[0])

def set_override(self, percentage, robot_number=1,
    slot_number=1):
    self.process_request(f'{robot_number};{slot_number};
        OVRD={percentage}')

def get_parameter(self, name, robot_number=1, slot_number
    =1):
    return self.process_request(f'{robot_number};{
        slot_number};PAR{name}')[1]

def set_parameter(self, name, value, robot_number=1,
    slot_number=1):
    self.process_request(f'{robot_number};{slot_number};PAW
        ={name};{value}')

def stop_immediately(self, robot_number=1, slot_number=1):
        self.process_request(f'{robot_number};{slot_number};
            STOP')
        self.turn_servos_off(robot_number, slot_number)

    def stop_at_next_cycle(self, robot_number=1, slot_number=1)
        :
        self.process_request(f'{robot_number};{slot_number};
            CSTOP')

    def is_running(self, robot_number=1, slot_number=1):
        run_status = int(self.process_request(f'{robot_number
            };{slot_number};STATE')[4][:2], 16)
        return bool(int(f'{run_status:08b}'[1]))

    def move_to_safe_position(self, robot_number=1, slot_number
        =1):
        if not self.is_running(robot_number, slot_number):
            self.turn_servos_on(robot_number, slot_number)
            self.process_request(f'{robot_number};{slot_number
                };MOVSP')
        else:
            raise RobotControllerError(ua.StatusCodes.
                BadInvalidState)

    def resume_program(self, program_name, repeated,
        robot_number=1, slot_number=1):
        if program_name not in self.program_list:
            raise RobotControllerError(ua.StatusCodes.
                BadInvalidArgument)
        if self.is_running(robot_number, slot_number):
            raise RobotControllerError(ua.StatusCodes.
                BadInvalidState)
        if self.get_current_program(robot_number, slot_number)
            != program_name:
            self.initialize_all_slots(robot_number, slot_number
                )
            self.process_request(f'{robot_number};{slot_number
                };PRGLOAD={program_name}')
        self.turn_servos_on(robot_number, slot_number)
        self.process_request(f'{robot_number};{slot_number};RUN
            {program_name};{int(not␣repeated)}')

    def reset_error(self, robot_number=1, slot_number=1):
        self.process_request(f'{robot_number};{slot_number};
            RSTALRM')

    def reset_program(self, robot_number=1, slot_number=1):
        if not self.is_running(robot_number, slot_number):
            self.process_request(f'{robot_number};{slot_number
                };RSTPRG')
        else:
            raise RobotControllerError(ua.StatusCodes.
                BadInvalidState)

    def turn_servos_on(self, robot_number=1, slot_number=1):
        self.process_request(f'{robot_number};{slot_number};
            SRVON')
        time.sleep(2)

    def turn_servos_off(self, robot_number=1, slot_number=1):
        self.process_request(f'{robot_number};{slot_number};
            SRVOFF', timeout_factor=20)

    def get_most_recent_error(self, robot_number=1, slot_number
        =1):
        return self.process_request(f'{robot_number};{
            slot_number};ERRORLOGTOP')[3]

    def setup_controller(self, robot_number=1, slot_number=1):
        while True:
            try:
                self.process_request(f'{robot_number};{
                    slot_number};CNTLON')
                break
            except RobotControllerError:
                time.sleep(1)


class RobotControllerError(Exception):
    def __init__(self, status_code, error_code=None):
        Exception.__init__(self)
        self.status_code = status_code
        self.error_code = error_code
```

Listing 7.4: robotcontrollerclient.py

```
import time
```

```
from telnetlib import Telnet
from threading import Thread, Lock


class TelnetClient:
    def __init__(self, ip, port, timeout, setup_method,
       test_command, encoding):
        self.ip = ip
        self.port = port
        self.timeout = timeout
        self.setup_method = setup_method
        self.test_command = test_command
        self.encoding = encoding
        self.client_lock = Lock()
        self.client = None
        self.connect_to_server()
        ConnectionMaintainer(self).start()

    def connect_to_server(self):
        while True:
            try:
                with self.client_lock:
                    self.client = Telnet(self.ip, self.port,
                       self.timeout)
                break
            except OSError:
                time.sleep(1)

    def send_request_and_return_response(self, req,
       timeout_factor=1):
        try:
            with self.client_lock:
                self.client.read_very_eager()
                self.client.write(req.encode(self.encoding))
                time.sleep(self.timeout * timeout_factor)
                return self.client.read_very_eager().decode(
                   self.encoding)
        except EOFError:
            return ''


class ConnectionMaintainer(Thread):
    def __init__(self, telnet_client):
        Thread.__init__(self)
        self.telnet_client = telnet_client

    def run(self):
        while True:
            if self.telnet_client.
                send_request_and_return_response(self.
                telnet_client.test_command) == '':
                 self.telnet_client.connect_to_server()
                 self.telnet_client.setup_method()
            time.sleep(10)
```

Listing 7.5: telnetclient.py

```
import sys

from IPy import IP

from opcuaserver import OpcUaServerForRobotController

MAX_PORT = 65535
MIN_PORT = 0
USAGE = f'USAGE:␣python␣{__file__}␣<IP_OS>␣<PORT_OS>␣<IP_RC>␣<
    PORT_RC>'
INCORRECT_IP = 'Please␣specify␣correct␣IP␣addresses!'
INCORRECT_PORT = f'Please␣specify␣ports␣between␣{MIN_PORT}␣and␣
    {MAX_PORT}!'


def check_port_number(port_number):
    if port_number < MIN_PORT or port_number > MAX_PORT:
        raise ValueError()


if len(sys.argv) != 5:
    sys.exit(USAGE)

try:
    ip_os = str(IP(sys.argv[1]))
    ip_rc = str(IP(sys.argv[3]))
except ValueError:
    sys.exit(INCORRECT_IP)

try:
    port_os = int(sys.argv[2])
    port_rc = int(sys.argv[4])
    check_port_number(port_os)
    check_port_number(port_rc)
except ValueError:
    sys.exit(INCORRECT_PORT)

OpcUaServerForRobotController(ip_os, port_os, ip_rc, port_rc).
    start()
```

Listing 7.6: startserver.py

7.3 Robot programs

```
1 Ovrd 20
2 Mov P15,40
3 Mov P50,40
4 Mov P14,40
5 Mov P10,40
6 Mov P16,40
7 End
8
```

Listing 7.7: prj.mb5

```
DEF POS P15=(230.03,0.00,378.00,0.00,0.00,90.00)(7,0) *// HOME
DEF POS P50=(238.63,-17.24,60.81,0.00,0.00,90.15)(7,0) *//
    Color Measurement
DEF POS P14=(234.60,-69.11,70.25,20.82,0.47,152.63)(7,0) *//
    Chute
DEF POS P10=(351.12,-14.42,57.68,0.00,0.00,90.00)(7,0) *//
    Assembly Socket High
DEF POS P16=(320.90,-30.91,62.72,0.00,0.00,90.00)(7,0) *//
    Search for Hole in Bottom
```

Listing 7.8: prj.pos

List of Figures

List of Tables

Bibliography

[1] F. Pauker, *OPC UA Information Model Design - Informationsmodellierung für Cyber-Physical Production Systems.* 2019.

[2] J. Rinaldi, *OPC UA : the everyman's guide to the most important communications architecture of industrial automation.* [North Charleston, SC]: [CreateSpace Independent Publishing Platform], 2016.

[3] H. Mersch, M. Schleipen, J. Aro, J. Bajorat, and C. Berger, *Praxishandbuch OPC UA : Grundlagen – Implementierung – Nachrüstung – Praxisbeispiele.* Würzburg: Vogel Business Media, 1. auflage. ed., 2018.

[4] "Soap," in *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*, pp. 83–114, Heidelberg: Spektrum Akademischer Verlag, 4 ed., 2010.

[5] M. Damm, S.-H. Leitner, and W. Mahnke, *OPC Unified Architecture.* Berlin, Heidelberg: Springer Berlin Heidelberg : Imprint: Springer, 2009.

[6] "Unified architecture." `https://opcfoundation.org/about/opc-technologies/opc-ua/`. last accessed on 2019/07/24.

[7] "Ua companion specifications." `https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/`. last accessed on 2019/06/30.

[8] "Opcfoundation/ua-nodeset/schema." `https://github.com/OPCFoundation/UA-Nodeset/tree/master/Schema`. last accessed on 2019/07/01.

[9] O. Foundation, *OPC Unified Architecture Part 100: Devices.* 2019.

[10] V. e.V., *OPC 40010-1 OPC UA for Robotics Part 1: Vertical Integration Edition 1.0.* 2019.

[11] L. Lo Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[12] "What's the difference between "brownfield" and "greenfield" iiot scenarios?." https://www.machinedesign.com/sites/machinedesign.com/files/0617_GreenBrown_PDFlayout.pdf. last accessed on 2019/07/22.

[13] "Python 3.7.3 documentation." https://docs.python.org/3/. last accessed on 2019/06/18.

[14] "Pycharm features." https://www.jetbrains.com/pycharm/features/. last accessed on 2019/06/18.

[15] "Reference." https://git-scm.com/docs. last accessed on 2019/07/01.

[16] "Python opc-ua documentation." https://python-opcua.readthedocs.io/en/latest/. last accessed on 2019/06/18.

[17] "Siemens opc ua modeling editor (siome) zur umsetzung von opc ua companion spezifikationen." https://support.industry.siemens.com/cs/document/109755133/siemens-opc-ua-modeling-editor-(siome)-zur-umsetzung-von-opc-ua-companion-spezifikationen?dti=0&lc=de-WW. last accessed on 2019/06/18.

[18] M. E. Corporation, *Connection with personal computer.* 2015.

[19] M. E. Corporation, *Mitsubishi Industrial Robot CR750/CR751/CR760 Series Controller INSTRUCTION MANUAL Troubleshooting.* 2017.

[20] "Uaexpert – ein vollausgestatteter opc ua client." https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uaexpert.html. last accessed on 2019/06/18.

[21] M. E. Corporation, *Mitsubishi Industrial Robot CR750/CR751 series controller Ethernet Function Instruction Manual.* 2014.

[22] "Bcm2837." https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md. last accessed on 2019/06/17.

[23] "Raspberry pi revision codes." https://www.raspberrypi.org/documentation/hardware/raspberrypi/revision-codes/README.md. last accessed on 2019/06/17.

[24] M. E. Corporation, *Mitsubishi Industrial Robot CR750-D/CR751-D/CR760-D Controller INSTRUCTION MANUAL Controller setup, basic operation, and maintenance.* 2017.

[25] M. E. Corporation, *Mitsubishi Industrial Robot CR750-D/CR751-D Controller RV-2F-D Series Standard Specifications Manual.* 2017.

[26] I. Ayatollahi, *Entwicklung eines OPC UA Servers für eine NC-Maschine.* 2014.