



Functional Safety Communication based on OPC UA

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

David Reitgruber

Matrikelnummer 11777716

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

Dipl.-Ing. Thomas Frühwirth

Wien, 10. Februar 2020

David Reitgruber

Wolfgang Kastner



Functional Safety Communication based on OPC UA

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

David Reitgruber

Registration Number 11777716

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

Dipl.-Ing. Thomas Frühwirth

Vienna, 10th February, 2020

David Reitgruber

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

David Reitgruber

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, 10. Februar 2020

David Reitgruber

Kurzfassung

OPC und sein Nachfolger OPC Unified Architecture (OPC UA) sind aus der industriellen Automation nicht mehr wegzudenken. Es wird verwendet um die immer steigende Komplexität der Systeme zu abstrahieren und zu modellieren. Hierzu wurde bisher ein Server/Client Modell verwendet, um die Kommunikation zwischen Systemen zu realisieren. In den letzten Jahren wurde OPC UA nun auch um eine Spezifikation für ein Publisher/Subscriber Kommunikationsmodell erweitert. Dies führte zusammen mit Time-Sensitive Networking (TSN) dazu, dass OPC UA auch für Echtzeitkommunikation geeignet ist [1]. Mit einer weiteren Spezifikation soll nun der funktional sichere Datenaustausch zwischen Systemen beschrieben werden. Das Dokument *OPC UA Part 15: Safety* beschreibt Protokolle und Services für den Datenaustausch mithilfe von OPC UA. Ziel dieser Arbeit ist es basierend auf dieser Spezifikation zwei sichere SPS-Systeme mittels OPC UA zu verbinden und sicherheits-spezifische Daten auszutauschen. Hierzu wird ein Beispielprogramm erstellt welches dann zum Testen und Evaluieren der Funktionalität verwendet wird.

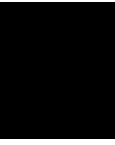
Abstract

OPC and its successor OPC Unified Architecture (OPC UA) are indispensable in the industrial automation. It gets used to abstract and model the evergrowing complexity of systems. So far a Server/Client model has been used to realize the communication between systems. Over the last years, OPC UA got extended with a specification describing the Publisher/Subscriber communication model. In addition to Time-Sensitive Networking (TSN), this resulted in OPC UA being able to implement real-time communication [1]. With an additional specification, functional safety data exchange shall be described. The document *OPC UA Part 15: Safety* shows protocols and services for data exchange with the help of OPC UA. The goal of this thesis is to connect two safe Programmable logic controller (PLC) systems via OPC UA and to enable the exchange of safety data based on the ideas of the specification. For this, an example program gets created, which is then used to test and evaluate the functionality.

Contents

Kurzfassung	vii
Abstract	ix
Contents	xi
1 Introduction	1
1.1 Motivation and problem statement	1
1.2 Methodical approach	2
1.3 Structure of this thesis	2
2 Technical background	3
2.1 OPC UA	3
2.2 OPC UA Standards	7
2.3 Best practice	8
2.4 Functional safety	9
3 OPC UA Safety	11
3.1 Communication structure	11
3.2 Information Model	13
3.3 Interfaces	15
3.4 Protocol	16
3.5 Relevant Standards	18
4 Prototype	21
4.1 B&R System	21
4.2 Implementation	22
4.3 Evaluation	28
5 Conclusion and future work	33
6 Appendix	35
6.1 Example Program	35
6.2 OPC UA method definition	36
	xi

6.3 Evaluation	36
List of Figures	39
List of Tables	40
Listings	40
Bibliography	41



Introduction

Since the third industrial revolution, automation systems have become more complex and more decentralized. This step towards Industry 4.0 resulted in widespread systems where sensors and actuators with different interfaces and communication standards were used. By that, a problem of interoperability occurred, which can be solved by using OPC UA. It allows us to make an abstract model of the different systems and connect them in a standardized way. Since the introduction of OPC UA, the release of different specifications expanded the functionality of the framework. One of the most recent ones being the Publisher/Subscriber specification which helped to take a big step towards real-time applications using OPC UA. A topic that was not covered in detail in any of those specifications until recently is functional safety. The latest released specification *OPC UA Part 15: Safety* describes services and protocols, which extend the framework to fulfill the requirements of functional safety as defined in the IEC 61508 and IEC 61784-3 standards. The goal of this thesis is to implement safe communication between two safe PLCs with the help of this specification.

1.1 Motivation and problem statement

Safety has always been an important aspect when it comes to automation systems and with the release of the new specification published by the OPC Foundation and the Profibus Nutzerorganisation E.V., the motivation was sparked to implement and test safe communication between two safe PLCs. OPC UA allows for interoperability and vendor-independent communication between automation systems and has therefore taken an important role in creating and easily connecting complex systems. Thus, an extension to the OPC UA mechanism which fulfills the functional safety requirements is of interest. The release of the specification presented a standardized way on how to model safe communication using OPC UA. This new document proposes not only the behavior of

the individual endpoints for safe communication but also how to create an information model to access these endpoints.

In this thesis, some of these techniques and models are used to create safe communication between two safe PLC systems. The idea is to read the safe inputs of one system, transfer them via safe communication to the other system, where they are then used to control the safe outputs.

1.2 Methodical approach

The first step is to study the OPC UA specification and OPC UA itself to get a general idea on how the implementation of safe communication and the creation of the needed information model works. Further, some research is done to find relevant standards regarding functional safety and safe communication. The next step is to create a simple application for the safe PLC systems, which is extended later on with the safe communication and used to test the information model. Next, the information model is created following the guidelines of the specification. When this is completed the application and the model are combined to create a system where the two PLCs can communicate with each other. Finally, the complete system is tested and evaluated and a review about the used services and protocols is given.

1.3 Structure of this thesis

Chapter 2 presents the technical background for this thesis. First, a closer look at OPC UA with special attention to information modeling is shown. Then, some information about the more relevant standards concerning this thesis and OPC UA is presented. At the end of this chapter, a few of the best practices concerning information modeling are listed.

Chapter 3 focuses on the specification *OPC UA Part 15: Safety*. The first section covers the general idea of the specification and gives a summary of the content. It shows the communication structure, information model, interfaces, and protocol used. After this, safety-relevant standards and their connection to *OPC UA Part 15: Safety* are presented.

Chapter 4 then shows the use of those best practices by creating an information model for the two PLCs such that the two systems can communicate with each other via a safe communication. At first, the system with all modules and periphery is described. This is followed by a short outline of the example program which is run on the PLCs. Then the implemented information model and the communication between the two PLCs are presented in more detail. The chapter concludes with an evaluation of the implemented example.

The thesis concludes with chapter 5, which gives a summary and an outlook for future work.

Technical background

2.1 OPC UA

The information and graphics for this chapter are taken from the book *OPC Unified Architecture* by Mahnke et al. [2]. Changes to graphics and information from other sources are marked.

OPC UA is a series of standards developed by the OPC Foundation. It is based on its predecessor the Classic OPC and extends its functionality. Not only does it describe the communication and data exchange between systems, but it also shows how to systematically model them. This model can be everything, from a simple sensor to an entire factory. The capability to do so, creates interoperability between systems and makes it relatively easy to connect them using the protocols described by OPC UA. These two parts, the transport and modeling of data, became the foundation of OPC UA as shown in Figure 2.1.

When it comes to transportation of data, OPC UA defines different mechanisms for different cases. An optimized binary TCP protocol is defined for intranet communication. For Internet communication mappings to standards like XML or HTTP are defined.

The OPC UA Meta Model provides rules on how to model information, it gives base modeling constructs, describes base types to build a type hierarchy, and defines entry points into the address space. The address space contains the different items and their properties described by the model. When accessing data, a user browses through the address space to find the desired information.

The OPC UA Services provide the interface between server and client, where the server acts as a supplier and the client as a consumer. The client can access small pieces of data without having to understand the complex model of the system.

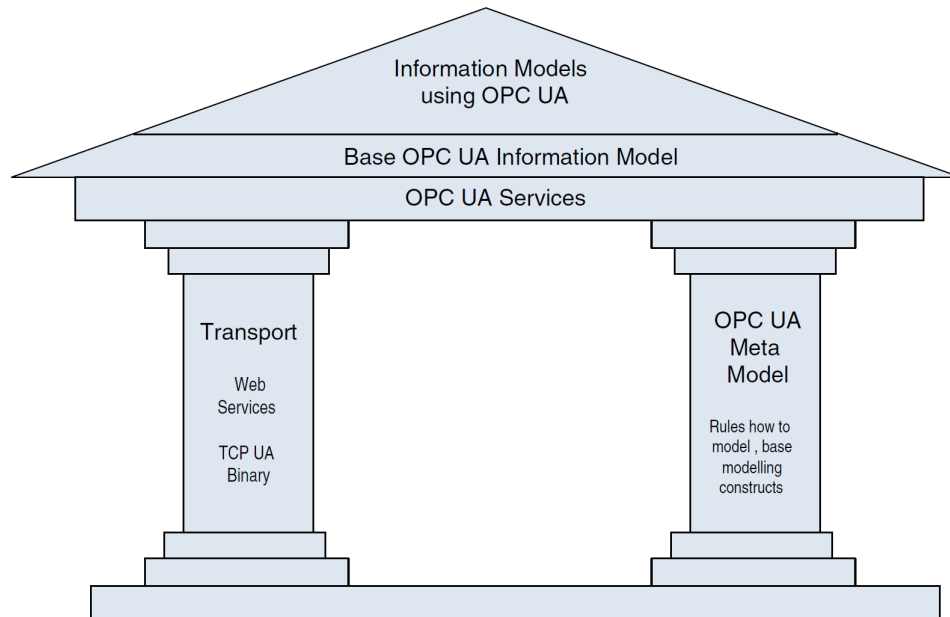


Figure 2.1: The foundation of OPC UA [2]

The architecture of OPC UA can contain different layers of information models as shown in Figure 2.2. Different organizations or vendors can introduce these layers by extending the base information model provided by OPC UA.

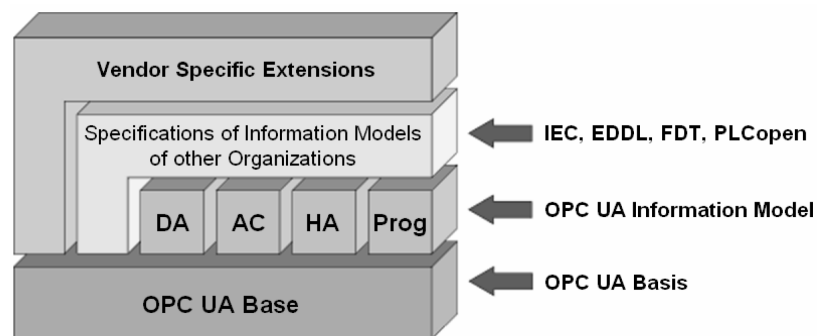


Figure 2.2: OPC UA layered architecture [2]

The elements of the graphical notation used in information modelling can be split into two parts: nodes and references (see Figures 2.3a and 2.3b).

Figure 2.4 shows an abstract representation of an information model. It displays references as a type of arrow, nodes as different types of shapes. References are used to connect nodes and to show different relationships between them. Depending on the purpose of a node, it can be of different node classes such as ObjectTypes, Methods, etc. A node

NodeClass	Graphical Representation
Object	Object
ObjectType	ObjectType
Variable	Variable
VariableType	VariableType
DataType	DataType
ReferenceType	ReferenceType
Method	Method
View	View

(a) Notation of Nodes

ReferenceType	Graphical Representation
Any symmetric ReferenceType	← ReferenceType →
Any asymmetric ReferenceType	— ReferenceType →
Any hierarchical ReferenceType	— ReferenceType →
HasComponent	— +
HasProperty	— +
HasTypeDefinition	— +
HasSubtype	← +
HasEventSource	— +

(b) Notation of references

Figure 2.3: OPC UA graphical notation (adapted from [2])

can contain attributes which can differ depending on the NodeClass. The attributes which all nodes have in common are NodeId, NodeClass, BrowseName, DisplayName, Description, WriteMask, and UserWriteMask. Of these seven attributes, the NodeId is the most important one, as it uniquely identifies a node. The client uses it to address nodes in a service call.

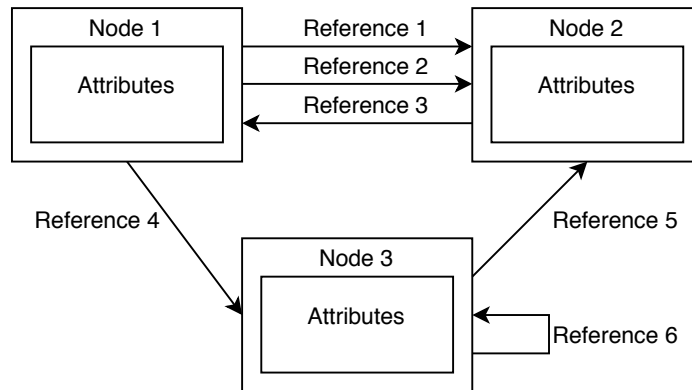


Figure 2.4: OPC UA simple model

To get a better understanding of how such a model could look like, this thesis presents a summary of the example given by Mahnke et al. in [2]. The modeled example is an air conditioner. It consists of a control module that offers information about the set temperature and humidity, the actual temperature and humidity, power consumption, fan speed, cooler state, and the possibility to turn the air conditioning on and off. The controller is connected to a temperature sensor, a humidity sensor, a wattmeter, a fan, and a cooler. Figure 2.5 shows a simple model of this example. The controller is an instance

of BaseObjectType and is connected to its sensors and actuators via a HasComponent reference. To turn the air conditioning on and off, two methods Start and Stop are used.

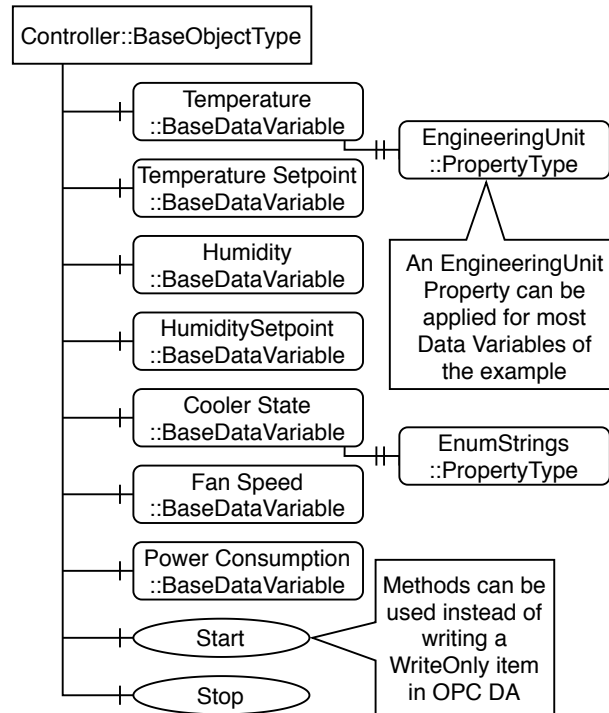


Figure 2.5: OPC UA simple model

To further show the power of OPC UA and the use of type hierarchies, the example is extended to allow for more than one controller as it would be in an office building where each room has air conditioning. By adding a furnace to the office building, we extend the example even further. The furnace controller contains information about the set temperature, the actual temperature, power consumption, gas consumption, burner state, and the possibility to turn it on and off. The controller is connected to a temperature sensor, a flow transmitter, a wattmeter, and a burner. Figure 2.6 shows the extended model with the use of type hierarchies. The ControllerType is used as an abstract base for all controllers. The temperature controller is an abstract type containing all common parts. FurnaceController and AirConitionerController are subtypes of TemperatureController and extend it with type-specific variables.

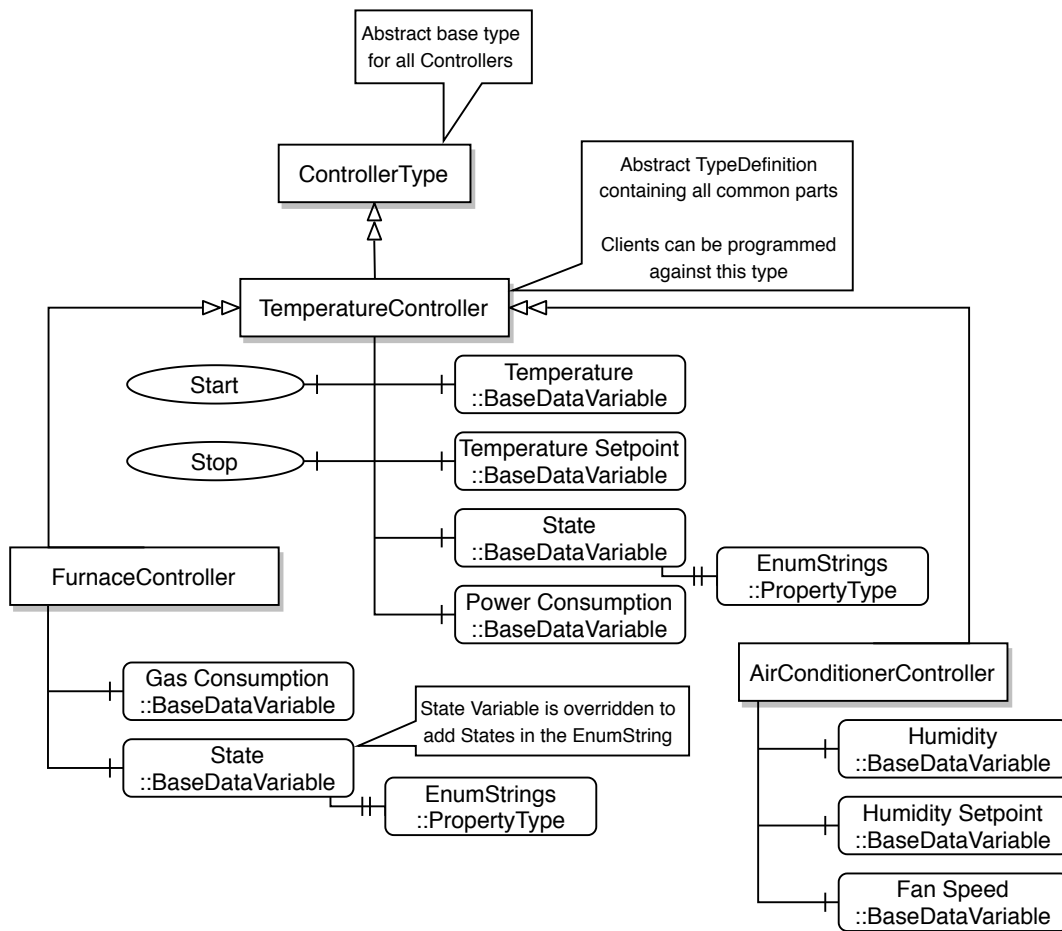


Figure 2.6: OPC UA extended model

2.2 OPC UA Standards

OPC UA was released as a 16 part series of standards under the name IEC 62541-x, where x stands for the specification number. Parts 1, 2, 11-14, and 100 are released under Edition 1, whereas Parts 3-10 are released under Edition 2. The newest specification part 15 was published recently, under version 1.04.

The more relevant parts for this thesis are 1, 2, 3, 4, 5, 6, 8, 14, 15.

- *Part 1 - Overview and Concepts* [3] gives an overview of the goals of OPC UA and provides an introduction to the concepts used to achieve these goals.
- *Part 2 - Security Model* [4] covers topics like user-authentication, access rights, and secure/encrypted message transmission.

- *Part 3 - Address Space Model* [5] gives a more in-depth description of address space within an OPC UA server.
- *Part 4 - Services* [6] provides information about the services the server provides and the client can call.
- *Part 5 - Information Model* [7] gives an in-depth description of how nodes, references, and the address space are used to create an information model.
- *Part 6 - Mappings* [8] contains information about data transfer between server and client, with more in-depth information about data encoding, secure communication, and transport protocols.
- *Part 8 - Data Access* [9] provides information about concepts of data access, address space organization, and behavior rules for data-types amongst other things.
- *Part 14 - PubSub* [10] describes the publish subscribe pattern communication model for OPC UA, which will be added to *Part 15 - Safety* [11] later on, since currently only Server/Client communication is mentioned.

Since *Part 15 - Safety* is the main motivation for this thesis, it is covered in more detail in Chapter 3.

2.3 Best practice

This section covers some of the best practices described by Mahnke et al. [2] to keep in mind when modeling concepts in OPC UA. Because OPC UA is very flexible concerning information modeling, the practices provided might not fit the desired application. Some general advice before going into detail is to model the server according to the application. There is no need for a big information model if the client is only accessing some variable values. Also, it might be better to use standardized types instead of creating new types. Clients that know such models can make use of them instead of generically accessing the data.

When it comes to structuring the information model, it can be useful to use objects, different reference types, and views. If clients are browsing the address space when accessing information, it is better to structure it with several objects instead of having one object with lots of variables. Different criteria can be used to group the variables to objects such as configuration data or measurement data. When creating a complex model where a simple hierarchy is no longer sufficient, it is good practice to make use of the different reference types, hierarchical or non-hierarchical. Before creating new reference types, it should be checked if already existing ones might fit the desired purpose. If a new one is created, it has to use the most appropriate reference type in the existing ReferenceTypes as supertype, since clients might filter based on standard ReferenceTypes when accessing the address space. If there is a big amount of nodes in the model, it might come in handy to use views to show only parts of the address space for each specific task.

2.4 Functional safety

With the growing complexity of industrial automation systems and the number of tasks they take over from humans, the safety of those systems is more relevant than ever. Safety is defined as the freedom from harm to people, property or the environment. Functional safety utilizes different methods to identify hazardous events that can cause harm and applies protective measures to reduce the consequences of them. To ensure the safety of a system, the principles of functional safety are used from the beginning of development until the retirement of the system.

An important step to achieve functional safety is risk assessment. A general method for this is described by Gehlen in [12], and consists of five steps:

- Establishing the limitations of a system
- Identifying hazards and hazard situations
- Assessing the risk for each identified hazard
- Evaluate the risk and deciding the need for risk reduction
- Removing or preventing the hazard with protective measures

The repeated execution of these five steps results in an iterative process, which is used to reduce hazards as much as possible and to find appropriate protective measures.

When it comes to analyzing and identifying the hazards and risks of a system, different methods can be used. Börcsök explains some of them in [13]. The forward-search starts with an event and follows the real timeline to find hazards that result from it. The opposite of this method is backwards-search. Here the consequences of already triggered events are used as the start for the analysis. It is searched in reversed chronological order for other events, which can result in the same consequences. Another method is the bottom-up search, where the chronological error propagation is examined. It starts with the origin error source and then each element of a system is analyzed to see if it is affected by it. The top-down search, as opposed to bottom-up, starts with the element that poses the most harm for people or the environment. From there possible error sources are searched, which finally leads to the root error cause.

Another important aspect to better assess the risk of a system or a process is likelihood-analysis, which estimates failure rates and frequency of different events. These rates are generally estimated through information won by experience. If a system is complex, most of the time there are not enough empirical values to create a statistic. In this case, fault-propagation models are used. With those models, the complex system is split up in smaller parts, which, concerning statistical evaluation, can be compared to other smaller elements. By combining the results of these subsystems, a failure probability of the whole system can be calculated. Some of these fault-propagation models are fault-tree, event-tree, block-diagram, and Markov chains.

After completing the risk assessment and calculating the failure probability, there are two ways a user can classify a system. One is by using the Safety Integrity Level (SIL) which is defined in the IEC 61508 [14]. Here, the system is categorized with a SIL of 1-4, depending on the average failure probability and how high the demand of the system is. Another way is the Performance Level (PL) defined in the ISO 13849-1 [15]. Here the system is classified with a PL from a-e, depending on the average failure probability.

Functional safety has to cover a large scope of applications and systems. Because of this, there exist a lot of standards for different areas covering this topic. One of the most important ones is the IEC 61508 [14], also referred to as the basic safety standard. It provides the basis for almost all safety applications. Covered in it is the complete lifecycle of safety-specific systems, starting at the development up to the retirement. Some other important standards are the IEC 61511 [16], which covers systems for the process industry sector, and the IEC 61784-3 [17], which covers functional safety communication. Also important are the ISO 13849 [15] [18], which covers safety-related parts of a control system, the ISO 12100 [19], which gives general design principles to create safe machines, and the IEC 62061 [20], which covers safety-related electrical, electronic and programmable electronic control systems. Section 3.5 presents more standards and how they connect to the specification *OPC UA Part 15: Safety*.

There exist many protocols for industrial Ethernet to enable two systems to communicate with each other, for example, PROFINET, POWERLINK, Modbus TCP or Ethernet. Some of these have implementations, which can ensure functional safety for communication. If a communication protocol is used, for which it is not possible to prove that it ensures functional safety, the *Black Channel Principle* can be used.

It was first described in the IEC 61508 [14] and allows for the transmission of non-safe and fail-safe data over the same network channel. Safety components utilizing this principle use an isolated safety protocol, which tunnels the non-safe network layer. Since this is only a protocol and not a physical connection, the available bandwidth is shared with the regular data exchange and depends on the used protocol. Because a non-safe channel is used, it is necessary to detect communication errors. To do so, a safety layer is implemented between the non-safe communication channel and the safety application, which performs the error checks. How accurately these communication errors need to be detected is defined by the SIL needed by the application. This *Black Channel Principle* is the basis for OPC UA Safety.

OPC UA Safety

The following chapter gives more in-depth information about the specification *OPC UA Part 15: Safety Release 1.04*. The content of this chapter is taken from [11]. It addresses the protocols and services used for data exchange and gives more detail on how to implement them. The specification defines a safety communication layer, which allows the exchange of safety-related data by devices using OPC UA. By implementing this communication correctly, it is possible to fulfill the requirements for the international standards IEC61508 and IEC61784-3. The protocol used performs different checks to ensure safe communication between two parties. These checks contain Cyclic Redundancy Check (CRC) to detect transmission errors, timeouts to make sure the communication is still up and running, monitoring numbers to check for the correct order of messages, and a set of IDs identifying messages.

3.1 Communication structure

To enable two systems to communicate with each other, OPC UA utilizes a Server/Client or the more recently introduced Publisher/Subscriber model and a TCP/IP or UDP connection. If both of these systems would run safety specific applications and they would exchange data, then this data would be transmitted over a non-safe channel. This means that on this channel, a communication error can occur, which could falsify or alter the data. Subsequently, the use of this modified data could lead to unexpected behavior in the receiver and cause problems with the safety application.

To prevent this from happening, the specification presents a way that both parties can safely communicate with each other over a non-safe channel. A safety transmission protocol is put on top of the standard OPC UA transmission system. This protocol is used to detect errors that might occur during the communication. Figure 3.1 shows the extended communication layers for OPC UA Safety.

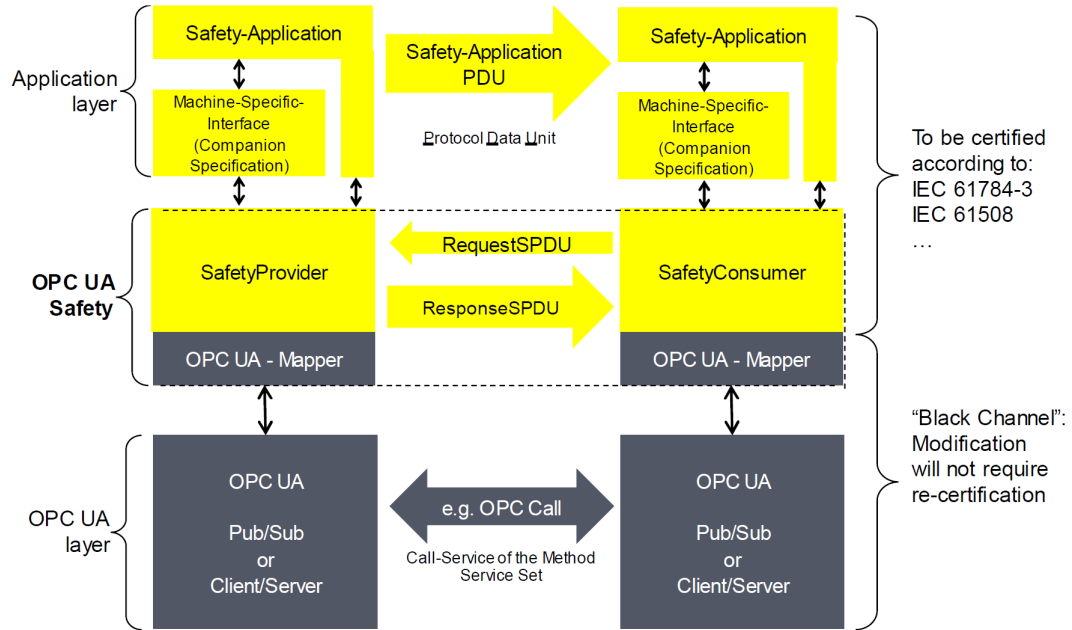


Figure 3.1: Safety layers [11]

The *OPC UA layer* implements the server and the client used for the communication. Here the method and service calls take place. The *application layer* contains the Safety-Application. The *OPC UA Safety layer* implements safe communication. It consists of the SafetyProvider and the SafetyConsumer. It is connected directly to the application layer through a machine-specific interface.

The communication between the SafetyProvider and the SafetyConsumer is based on a request/response schema and uses a point-to-point communication model to transmit the data unidirectional. Each request/response contains a Safety Protocol Data Unit (SPDU), which holds the safety data. But before they can exchange data, basic information, such as the IDs, is exchanged to ensure safe communication. After this initialization process, data exchange can be performed. If the application needs bidirectional communication, two pairs of SafetyProvider and SafetyConsumer have to be instantiated. It is also possible to create a multicast communication link. For each SafetyConsumer, its own SafetyProvider has to be created. Because the SafetyProvider requires low memory and the SafetyConsumer performs all the checks to ensure no errors occurred, it is possible to create multiple SafetyProviders on the same controller.

3.2 Information Model

Figure 3.2 shows the information model for the SafetyProvider. The SafetyProvider is a subtype of SafetyObjects and holds three components.

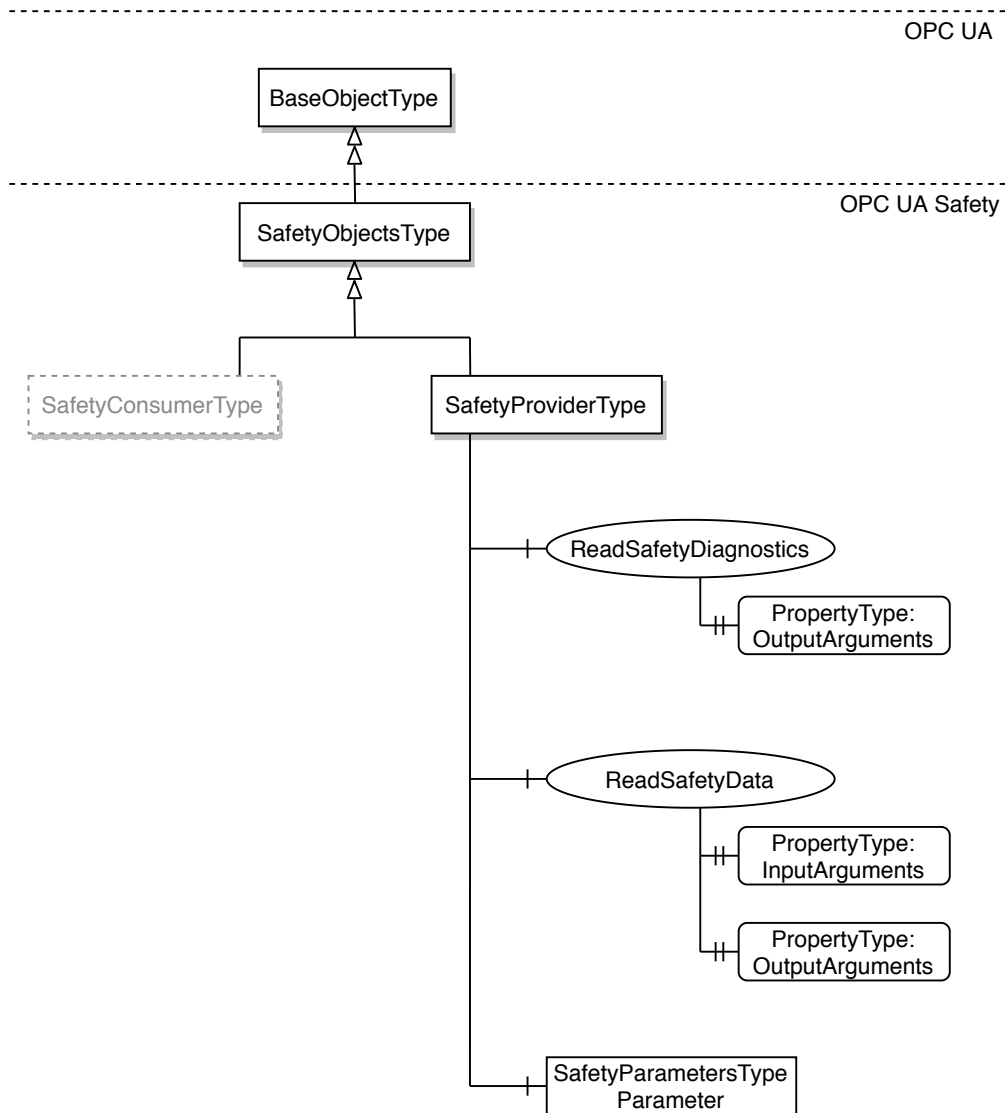


Figure 3.2: Server Objects for OPC UA Safety

One of the components is the SafetyParameters for which Figure 3.3 displays the information model. They hold important information like the SafetyProviderLevel or the SafetyProviderID and SafetyBaseID which are exchanged at the beginning of communication and determine if fail-safe or application data is transmitted. If an application uses the fail-safe data, it changes into a fail-safe state. The system is now in a state where no

or minimal harm can be caused to the system itself or people.

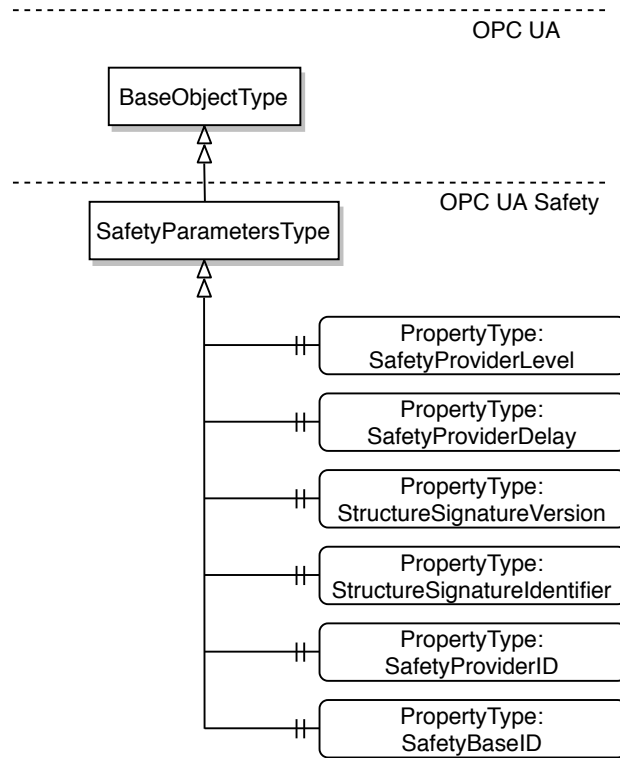


Figure 3.3: Safety Parameters for SafetyProvider

The method `ReadSafetyData` is used to read safety data from the `SafetyProvider`. Table 3.1 shows the signature of the method. The arguments `OutSafetyData` and `OutNonSafetyData` are application dependent and can be defined as needed.

The method `ReadSafetyDiagnostics` is used as a diagnostic interface for each `SafetyProvider` and can access the received and sent SPDUs. Table 3.2 displays the signature of the method.

Type	Datatype	Name
in	UInt32	InSafetyConsumerID
in	UInt32	InMonitoringNumber
in	Byte	InFlags
out	Structure	OutSafetyData
out	Byte	OutFlags
out	UInt32	OutSPDU_ID_1
out	UInt32	OutSPDU_ID_2
out	UInt32	OutSPDU_ID_3
out	UInt32	OutSafetyConsumerID
out	UInt32	OutMonitoringNumber
out	UInt32	OutCRC
out	Structure	OutNonSafetyData

Table 3.1: Signatur of ReadSafetyData

Type	Datatype	Name
out	UInt32	InSafetyConsumerID
out	UInt32	InMonitoringNumber
out	Byte	InFlags
out	Structure	OutSafetyData
out	Byte	OutFlags
out	UInt32	OutSPDU_ID_1
out	UInt32	OutSPDU_ID_2
out	UInt32	OutSPDU_ID_3
out	UInt32	OutSafetyConsumerID
out	UInt32	OutMonitoringNumber
out	UInt32	OutCRC
out	Structure	OutNonSafetyData

Table 3.2: Signatur of ReadSafetyDiagnostics

3.3 Interfaces

OPC UA Safety uses four interfaces to interact with the communication layers. These interfaces are described as abstract and informative. The correct implementation can change vendor-specific. The four interfaces are:

- Safety Application Program Interface (SAPI) is accessed during runtime to exchange safety data
- Safety Parameter Interface (SPI) is accessed during commissioning to set the safety parameters

- Diagnostic Interface (DI) is accessed during runtime to run diagnostics on the communication
- OPC UA platform interface (OPC UA PI) is accessed during runtime and connects the Safety Communication Layer to the non-safe OPC UA stack

3.4 Protocol

The protocol is based on a request/response schema. When the SafetyConsumer calls the ReadSafetyData method, the three input arguments specified in the signature of the method (see Table 3.1) are passed, which make up the RequestSPDU. The SafetyProvider then creates the ResponseSPDU according to the signature, by copying the SafetyConsumerID and the MonitoringNumber from the RequestSPDU directly to the ResponseSPDU. The calculation of the SPDU_ID is shown in Table 3.3.

$$\begin{array}{l}
 \text{SPDU_ID_1} := \text{SafetyBaseID (bytes 0..3)} \mathbf{XOR} \text{SafetyProviderLevel_ID} \\
 \text{SPDU_ID_2} := \text{SafetyBaseID (bytes 4..7)} \mathbf{XOR} \text{SafetyStructureSignature} \\
 \text{SPDU_ID_3} := \text{SafetyBaseID (bytes 8..11)} \mathbf{XOR} \text{SafetyBaseID (bytes 12..15)} \\
 \mathbf{XOR} \text{SafetyProviderID}
 \end{array}$$

Table 3.3: Calculation of SPDU_ID

The CRC_SPDU sequence created for the CRC calculation is made up of two parts, the STrailer, which consists of SafetyConsumerID, MonitoringNumber, SPDU_ID_3, SPDU_ID_2, SPDU_ID_1, Flags (in this order), and the SData, which holds the safety data. The data is encoded using big-endian. The calculation of the CRC is done in reverse order, starting by the highest byte and ending at byte 0. It is important to note that the NonSafetyData is not part of the CRC calculation.

The two state diagrams shown in Figure 3.4 and 3.5 describe, how the SafetyProvider and SafetyConsumer handle the protocol.

Because the SafetyConsumer performs all of the safety checks, the state diagram for the SafetyProvider is kept relatively simple and consists of only three states and three transitions. In the initialization state, the necessary data gets initialized with 0. After the initialization, it transitions into the state S1_WaitForRequest. There it waits for a request of the SafetyConsumer. When receiving such a request, it fetches the RequestSPDU, sets the MonitoringNumber and SafetyConsumerID in the SAPI and transitions into the state S2_PreparesSPDU. There the required flags are set, and the ResponseSPDU is built as described before. When completing this, the SafetyProvider sets the ResponseSPDU and transitions back into state S1_WaitForRequest.

The state diagram of the SafetyConsumer consists of 9 states and 15 transitions between them. It performs all the safety checks here, consisting of the CRC check, watchdog timeout, SafetyErrorIntervalTimer expired, and SPDU check. First, all the required data gets initialized to 0 in the initialization state. After this, it transitions

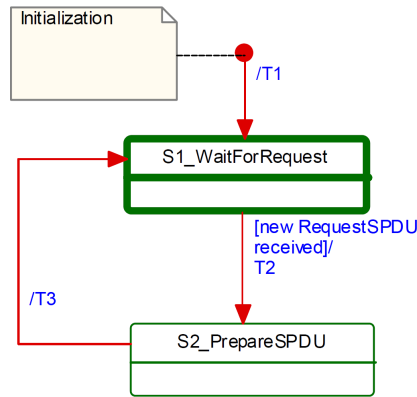


Figure 3.4: State Diagram for SafetyProvider [11]

into the S11_WaitForRestart state, where the SafetyConsumer waits until the Safety Layer is (re)started. When the SAPI.Enable is set to 1 it switches into the next state S12_Initialize_MNR, where a MonitoringNumber gets chosen, either to a previous one if known, or a random one within the allowed range. With the next transition into state S13_PrepareRequest, the SafetyConsumer enters the Request/Response part. Here the RequestSPDU is built and with the transition into state S14_WaitForChangedSPDU the RequestSPDU is sent. In this state, the Safety Layer is waiting on the ResponseSPDU from the SafetyProvider. If no new response is received, the SafetyConsumer changes into the S17_Error state. Otherwise, it transitions into S15_CRCCheckSPDU, where it performs the CRC check. If the CRC is erroneous and the SafetyErrorIntervalTimer has not expired, it changes into S17_Error. If the CRC is erroneous and the SafetyErrorIntervalTimer has expired, the SafetyConsumer switches back into the S13_PrepareRequest state. Otherwise, if the CRC check is ok, it transitions into S16_CheckResponseSPDU. Here the SafetyConsumerID, the SPDU_ID, and the MNR are checked. If those checks are not ok and the SafetyErrorIntervalTimer has not expired, it changes into S17_Error. If those checks are not ok and the SafetyErrorIntervalTimer has expired, it switches back into S13_PrepareRequest. Otherwise, if all checks are ok, it transitions into S18_ProvideSafetyData. There the safety data is provided to the application. From there it either switches to S11_WaitForRestart or S13_PrepareRequest, depending on if SAPI.Enable is set to 0 or 1. If the SafetyConsumer enters state S17_Error the Safety Data is set to the fail-safe values and the bit for operator acknowledgment is set.

This is just a summary of how the SafetyProvider and SafetyConsumer implement this protocol. The specification provides more detail on all guard conditions and activities performed in each state (see [11]).

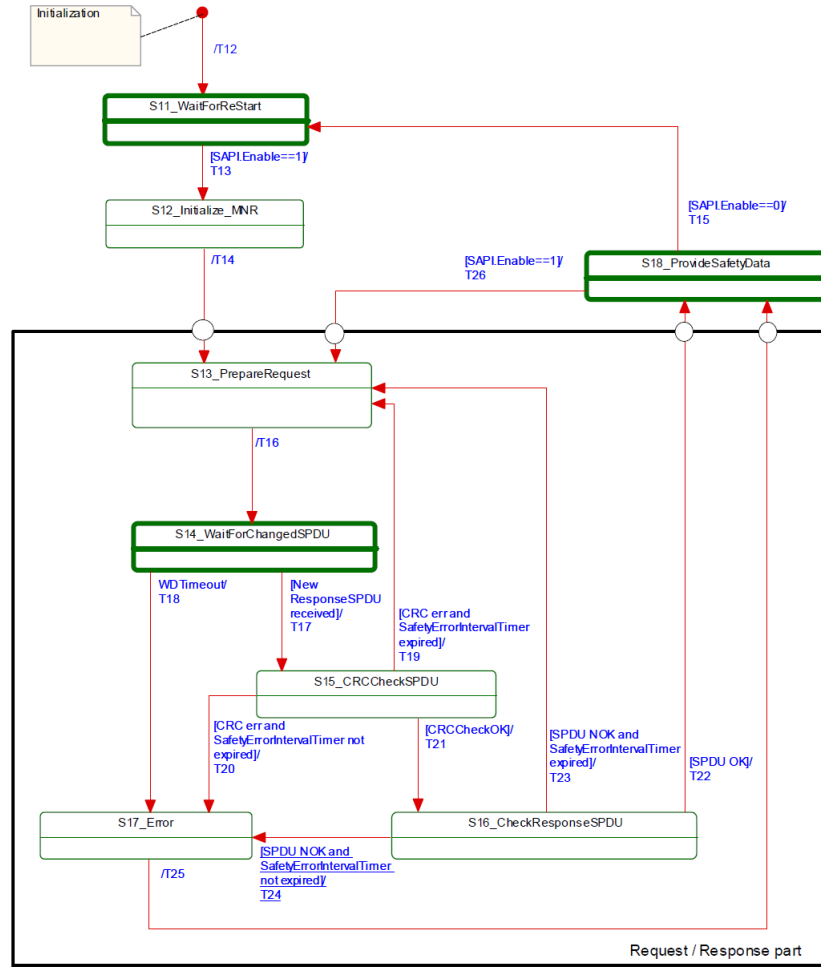


Figure 3.5: State Diagram for SafetyConsumer [11]

3.5 Relevant Standards

For this specification, several standards are important. As mentioned before in Chapter 2.2, the OPC UA specifications parts 1, 2, 3, 4, 5, 6, 8, and 14 are important. Further referenced in the specification are:

- *IEC 61784-3:2017* [17] which covers general rules and profile definitions for functional safety fieldbuses
- *IEC 61000-6-7* [21] which gives information about immunity requirements for parts intended to perform in safety systems in the industry
- *IEC 61508* [14] which provides information about the functional safety of electronic safety-related systems

- *IEC 61511* [16] which covers functional safety for the process industry sector
- *IEC 62061* [20] which describes the functional safety of safety-related electronic control systems
- *ISO 13849-1:2015* [15] which provides general principles for the design of safety-related parts of the control system
- *ISO 13849-2:2012* [18] which covers the validation of safety-related parts of the control system

Figure 3.6 shows the relation between those standards and the specification. An arrow from one standard to another means that it gets referenced in that standard.

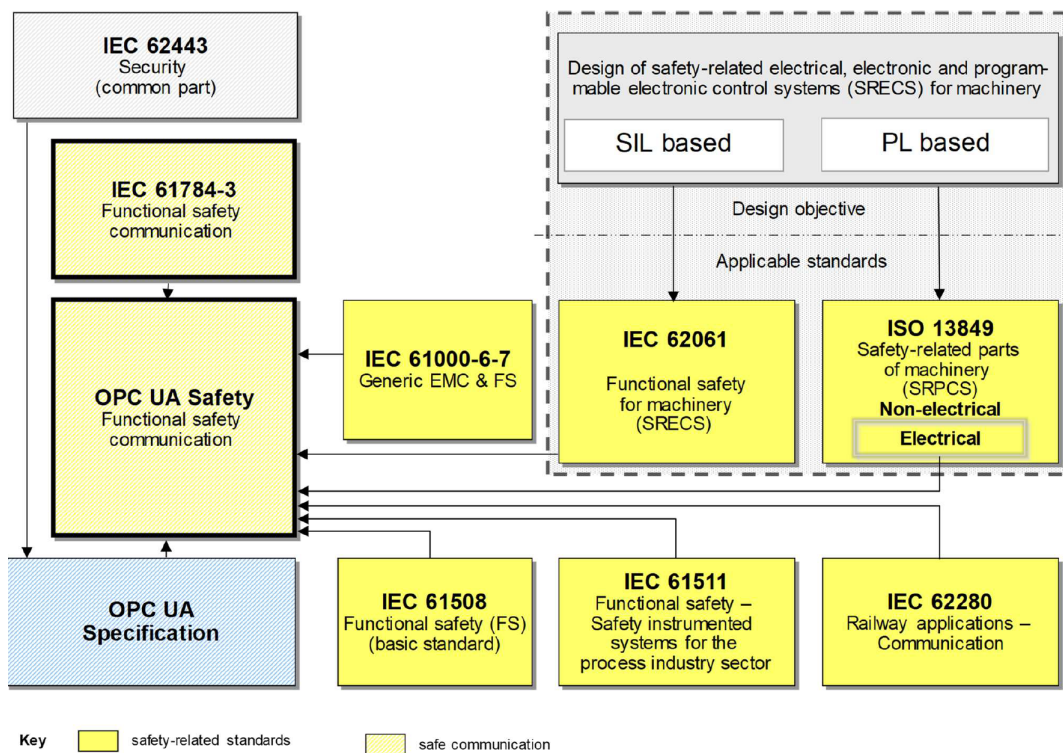


Figure 3.6: Relationship between standards [11]

Prototype

This chapter presents the implementation of safe communication between two safe PLCs with regards to the specification. It takes a closer look at the used systems and how to use OPC UA on them, before describing the example program run on them. After this, the main motivation of this thesis is shown, the implementation of safe communication between the two PLCs. For this, the information model presented in chapter 3 is used, and the example program is extended to incorporate the OPC UA connection and the fail-safe states. Further, this chapter describes some of the pitfalls when operating OPC UA on the B&R PLCs. It closes with an evaluation of the implementation.

4.1 B&R System

The system used in this thesis is produced by the company B&R, a manufacturer of automation technology. It's an ETA system with the following elements: a PLC with its modules, an input field with a button, a rotary switch, and an emergency stop button, a light curtain, a fan, the necessary cabling and control devices. Table 4.1 shows the modules and elements of the PLCs.

Module	Identifier
CPU	X20CP1586
SafeLogic	X20SL8100
Bus Coupler	X20BC0083
Power Supply	X20PS9400
Safe Digital Output	X20SO2110
Digital Input	X20DI6371
Safe Digital Input	X20SI4100

Table 4.1: Components of the PLC

Figure 4.2 displays the configuration of this hardware. The order of the components corresponds to Table 4.1. Connected to one of the safe input modules are the emergency stop button and the light curtain, which uses 2-channel equivalent evaluation to detect errors. The rotary switch connects to the other module, where each valid state connects to a standard safe digital input. The button connects to unsafe inputs. The fan connects to the safe outputs via a contactor. Figure 4.2 shows this configuration.

Figure 4.1 displays the system setup showing only the emergency stop button and the fan from the in/outputs.

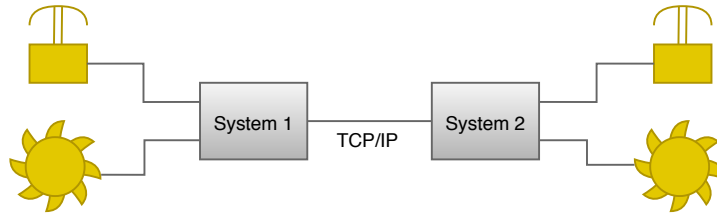


Figure 4.1: System setup

4.2 Implementation

4.2.1 Example Program

As a basis to test safe communication, an example program provided by B&R in [22] is used and the information and graphics for this section are taken from this manual. The project as provided was not finished since some of the safety aspects were not implemented. And to make it more convenient to work with, the project was rewritten from structured text to ANSI C (see Listing 6.1). The example simulates the control of a circular saw. The green start button starts and stops the saw. This button also acknowledges errors and fail states. The light curtain prevents from reaching into the saw and will stop it if the light curtain gets interrupted. The emergency stop button stops the saw when pressed and keeps it stopped until the button is unlocked again. The rotary switch changes between automatic and manual mode, which are described later on. Figure 4.2 shows the hardware setup of this example. There are two differences between the shown setup and the real hardware used: a fan simulates the saw and the rotary switch only has two valid states and not 8 as shown.

The program implements the safety requirements as described in [22]. After startup no acknowledging of the emergency stop button, the light curtain or rotary switch is necessary. An acknowledgment is required if:

- the emergency stop button gets unlocked
- the light curtain was interrupted
- the rotary switch is in an invalid state for longer than 250 ms

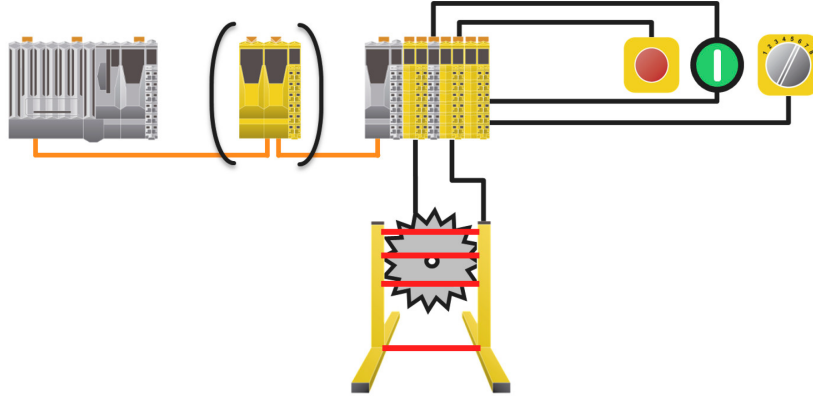


Figure 4.2: Hardware setup of the Example Project [22]

The simultaneity of the multi-channel evaluation for the emergency stop button must lie within 200ms, for the light curtain within 300ms.

As described before, the application has two operating modes, which can be chosen by the rotary switch: a manual and an automatic mode. In the manual mode, the light curtain is not active, the output is only active as long as the button is pressed and a violation of the safety requirements needs to be acknowledged with a positive pulse on the button. In the automatic mode, the light curtain is active, the output is always set as long as no safety violation occurs and such a violation needs to be acknowledged with a positive pulse on the button.

The main logic of the program is developed in Automation Studio (AS) [23] and written in ANSI C. To evaluate the safety elements of the application, the SafeDesigner (SD) [24], an application developed by B&R to program safe PLCs, was used and the logic was implemented as a Function block diagram (FBD). The connection between those two programs is made in the SD in the Safe View Tab. There it is possible to assign variables used in AS to variables used in the SD. Figure 4.3 shows an example of such an assignment.

ID	Name	Variable	Type	CPU Variable	Slot	Description
SL1						SafeLOGIC ID 1
SL1.SM1	X20SL8100				IF3.ST1	X20SL8100 X20 SafeLOGIC, PO...
SL1.SM2	X20SO2110				IF3.ST2.IF1.ST2	X20SO2110 X20 Safe Digital Out...
SL1.SM3	X20SI4100				IF3.ST2.IF1.ST4	X20SI4100 X20 Safe Digital In, 4...
General						
SafeModuleOk			SAFEBOOL			
SafeDigitalInput01			SAFEBOOL			24 VDC, sink
SafeDigitalInput02			SAFEBOOL			24 VDC, sink
SafeDigitalInput03			SAFEBOOL			24 VDC, sink
SafeDigitalInput04			SAFEBOOL			24 VDC, sink
SafeEquivalentInput0102		sdiStateLightCur...	SAFEBOOL	sawcdiStateLightCurtain		2-channel evaluation equivalen...
SafeAntivalentInput0102			SAFEBOOL			2-channel evaluation antivalent...
SafeEquivalentInput0304		sdiEStop	SAFEBOOL	sawcdiStateEStop		2-channel evaluation equivalen...
SafeAntivalentInput0304			SAFEBOOL			2-channel evaluation antivalent...
SafeChannelOK01			SAFEBOOL			Status digital input (1=OK)

Figure 4.3: Assignment of variables in the Safe View

It is important to note, that it is possible to configure the safe digital outputs with a restart lock. This is done in the module configuration by setting the digital output to direct as shown in Figure 4.4. If this configuration is chosen, the output needs to be released with a delay. In this case, a delay of 100ms is chosen, which is achieved in the SD with a timer.







Name	Value	Unit	Description
 X20SO2110			
 Function model	default		Module operating mode
 General			
 Output signal path			
 DigitalOutput01	direct		
 DigitalOutput02	direct		

Figure 4.4: Configuration of output module

In the Safe View Tab, the parameters for each module can be set. For most of the inputs and outputs used, the default values can be applied. An exception to this are the inputs for the light curtain. Each input channel is assigned a pulse mode which is used to choose between the internal or external clock generation. Since the light curtain is an active sensor, this signal needs to be deactivated for it to work correctly. Figure 4.5 displays this configuration for SafeDigitalInput01.

Model no.:	X20S14100	
Description:	X20 Safe Digital In, 4xl, 24V	
SafeMODULE ID:	3	
Import file:	-	
Parameter	Value	Unit
Safety Response Time		
Manual Configuration	No	
Safe Data Duration	20000	us
Additional Tolerated Pac	0	packets
Packets per Node Guar	5	packets
SafeDigitalInput01		
Pulse Mode	no Pulse	
Filter Off	200000	us
Filter On	200000	us
Discrepancy Time	50000	us
SafeDigitalInput02		
Pulse Source	default	
Pulse Mode	no Pulse	
Filter Off	200000	us
Filter On	200000	us
SafeDigitalInput03		

Figure 4.5: Input Configuration for the Light Curtain

4.2.2 OPC UA in Automation Studio

Configuration

Recent B&R PLCs are equipped with an OPC UA System, which can be activated in the CPU configuration. After activating it, it is possible to access the settings of the system. Here the port the system operates on can be set, the user authentication changed and some general limits and settings of the OPC UA server set. An important setting to change, when using an external information model, is the PV Model version of the nodeset files. It is set to version 1.00 by default, but it needs to be set to 2.00 for it to work with imported nodeset files. Such a file describes the information model using XML.

The information model used is specified in chapter 3 and was implemented in UA Modeler [25]. Figure 4.6 shows the OPC UA diagram of the SafetyProvider, Figure 4.7 of the SafetyParameters implemented in UA Modeler.

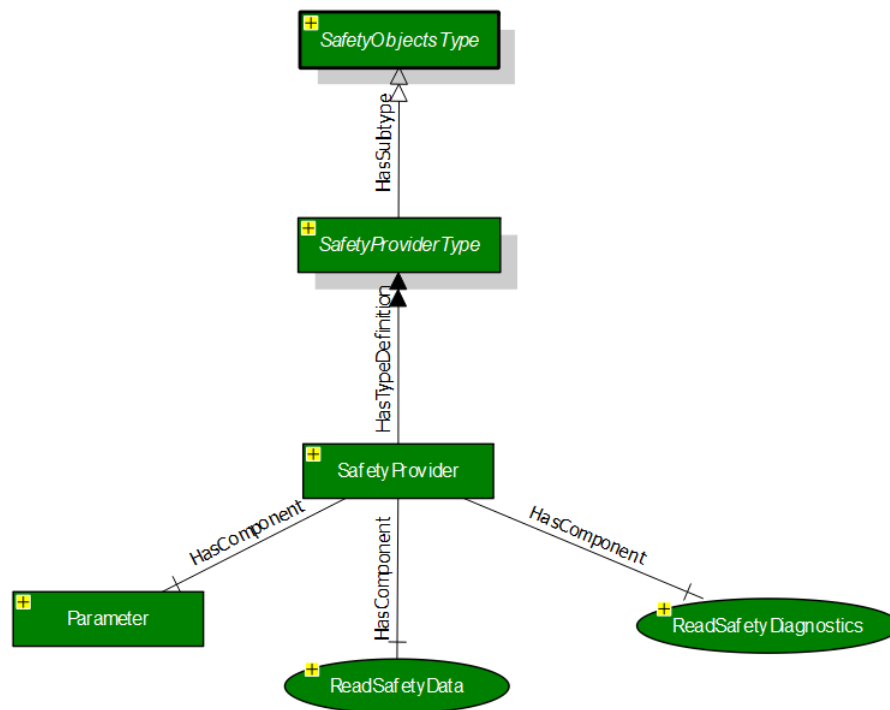


Figure 4.6: SafetyProvider model in UA Modeller

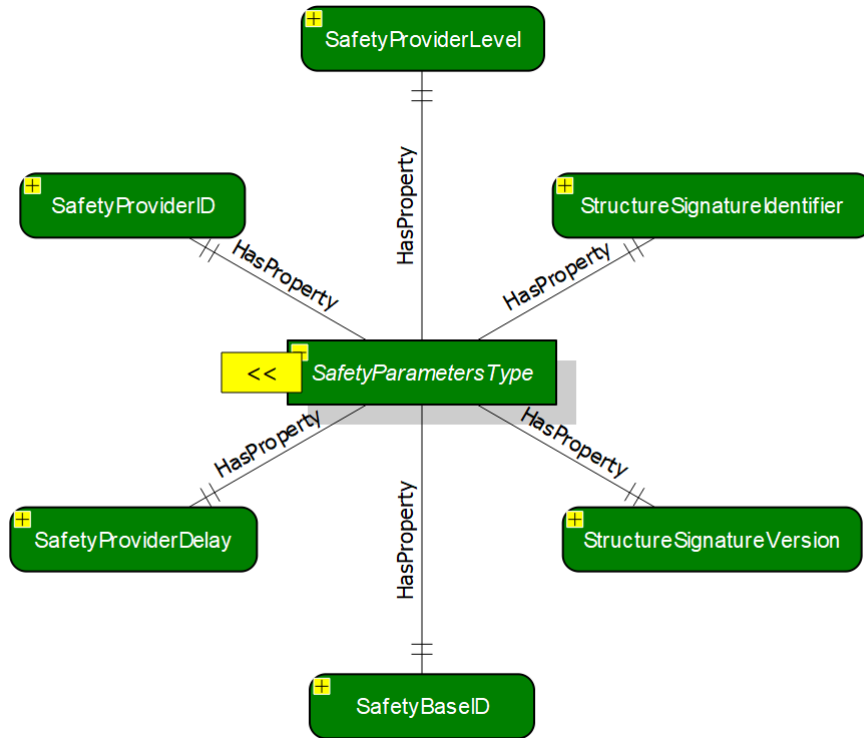


Figure 4.7: SafetyParameter model in UA Modeller

To use it in AS, the model has to be exported as an XML file, then it has to be copied into the OPC UA folder of the configuration in AS and the file ending needs to be changed to *.uanodeset*. Another thing to consider is the version of the integrated models, since with UA Modeller by default newer, unsupported versions are used. The supported versions are shown in Table 4.2.

NamespaceUri	Version	PublicationDate
http://opcfoundation.org/UA/	1.04	2018-05-15T00:00:00Z
http://opcfoundation.org/UA/DI/	1.01	2013-12-02T00:00:00Z
http://PLCopen.org/OpcUa/IEC61131-3/	1.00	2010-03-24T00:00:00Z

Table 4.2: Supported Versions on B&R Systems

To use the methods described in the model, they need to be declared in AS. For this, the method with its input and output arguments needs to be defined as an OPC UA method in the *.uam* file of the project. Listing 6.2 shows this for the *ReadSafetyData* method.

The declared methods are then visible in the default view of the OPC UA system. To connect the information model with the methods in AS the references need to be set. This is done in the properties of a method in the default view. There the namespace

and index of the method in the nodeset file are provided. Figure 4.8 shows this for the ReadSafetyData method.







Eigenschaften - ReadSafetyData			
Name		Wert	Wert
 ReadSafetyData			
 Autorisierung			
 Knotenbeschreibung			
 Referenz			
 Implementierung von		ns=5;i=7003	
 Attribute Maske Schreiben			

Figure 4.8: Reference for ReadSafetyData

Server & Client

Since this work requires a bidirectional communication between the two PLCs, it is necessary that each PLC is both, Server and Client. AS contains predefined libraries that can be used to operate the OPC UA server and communicate with it.

The *AsOpcUas* library provides interfaces to the OPC UA server. It contains a few different functions, such as creating and deleting methods or getting a namespace index. This work only needs the *UaSrv_MethodOperate* function. Since this function works asynchronously it needs to be called cyclically. This function checks, whether a method is called and once called to execute it. Figure 4.9 shows the states of this function and thus of the server. In the first state the server checks, whether a client called the method. If this is the case, valid data is available on the input arguments. The server then switches into the next state where it executes the method. When this is done, it changes into the last state and finishes up the method call. Now the output variables hold valid values for the output arguments of the method, and they are transferred to the client.

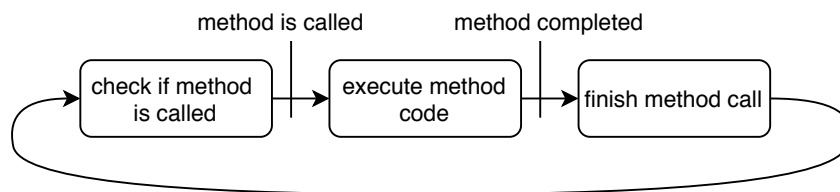


Figure 4.9: States of the Server

The *AsOpcUac* library supports the communication with the OPC UA server. It contains different functions like connect/disconnect to a server, subscribe to a server or write a node. This thesis only needs a few of those functions. Figure 4.10 shows the states the client completes, where each state contains a function call to a library function. All of those functions are asynchronous and must be called cyclically until the Done variable of the function is set to TRUE.

The first step of the client is to connect to the OPC UA server. Here it provides the username and password, if necessary, and the IP-address and port. This call returns a connection handle, which the client then uses in the following function calls. If the connect was successful, the next step is to obtain the namespace index of the PLC with the `getNamespaceIndex` function. This index is used to get the method handle. The handle represents the node for further use in other functions, such as the method call. After obtaining the handle, the program is ready to call the method. Here it uses the connection handle and method handle to call the method on the server. This is done cyclically until the program exits. On exit, it releases the method handle and the client disconnects from the server.

If the connection between server and client is lost because a function timed out, which can be caused by an overloaded server or a disconnected cable, then the client automatically tries to reconnect to the server and get the same session as before. If this is not successful then the connection must be reestablished as shown in Figure 4.10 starting in the connect state.

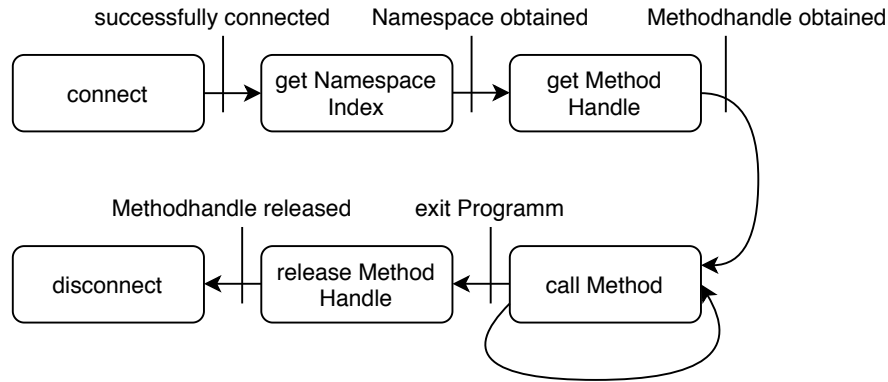


Figure 4.10: States of the Client

4.3 Evaluation

To evaluate the functionality of the program and the connection between the PLCs a simple example of safe communication was implemented. The goal for this example is, to read the emergency stop button from the safe inputs from one PLC, transfer it with OPC UA to the other PLC and use it there to control a safe output. The method used for the data exchange is the `ReadSafetyData` method described in the specification and used in the information model. For simplicity, the `SafetyData` variable was changed from a struct to a bool. Another functionality shall be to detect connection errors, in which case the PLC should go into a fail-safe state. This is implemented by setting the method timeout value in the application. In Figure 4.11 this test setup can be seen. In addition to the two PLCs, it also shows the stress tester and Wireshark connection, which are used later on.

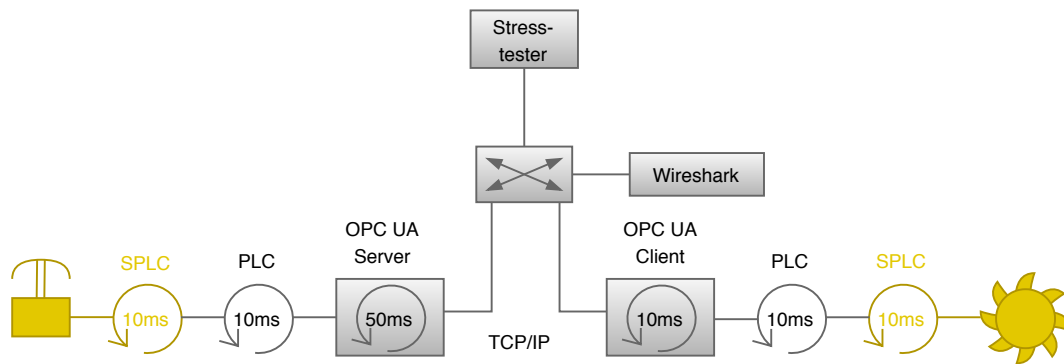


Figure 4.11: Test setup

4.3.1 Timing Analysis

Since this example uses a Server/Client model with a TCP/IP connection, the client needs to send a request each time it wants to read the value of the emergency stop button. As soon as the initial connection between the server and the client is established, the client starts sending requests to the server. It sends a new one, as soon as the server responds or every 10ms if the server responds faster than that. In our case, the server response takes about 50ms. This stems from the 10ms cycle time of the PLC and the five states of the state machine the server has to get through until it sends the response. This results in a worst-case Round Trip Time (RTT) of 80ms, which consists of 10ms until the client sends the request, 60ms until the server responds, and 10ms until the client reads the received value. This time is then used to set the timeout of the method call, which is used to detect a connection error. This could be a disconnected cable, or an overloaded server, which can not reply in time. The client then uses the received value, or the fail-safe value in case of an error, in the program logic to control the fan connected to the safe output. Figure 4.12 shows the requests and responses of this communication in a sequence diagram.

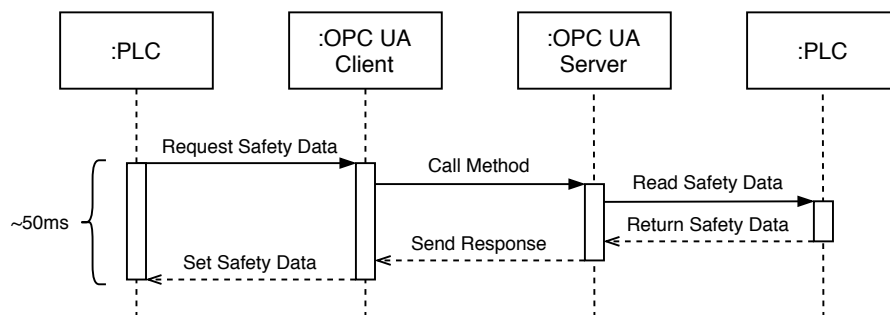


Figure 4.12: Sequence diagram of the communication

The first test performed used different values of timeouts for the method call, namely 40ms, 60ms, and 80ms. In the first case, every request resulted in a timeout as expected

and the application is always in a safe state. With 60ms, most of the requests got a response and the application performed correctly, but still, sometimes a timeout occurred. This depends on when within the cycle one PLC sends the request and when the other PLC receives this request. That may put the RTT below or above the timeout value. This problem disappeared almost completely when the 80ms timeout was used, since the worst-case RTT is 80ms. But the stress test, later on, will show, that with 80ms problems can still occur.

That behavior changes when in addition to the two PLCs, we sent requests from external systems to the OPC UA server. Because the server needs to perform other methods or read other attributes apart from the safety application, the response time of it for the ReadSafetyData method call may get greater than 50ms, which would result in a greater worst-case RTT. Thus, a timeout could also occur with 80ms. A simple test to show this behavior is to use UA Expert [26] to call the ReadSafetyData method. When doing so, the server queries the method call and then executes it. While this is executed, the PLC calls the ReadSafetyData method, which is also queried and now needs to wait for the completion of the UA Expert call. This results in a 50ms wait, which in further consequence results in a RTT of the PLC request greater than 100ms, and therefore the method times out and the application switches into the safe state.

4.3.2 Stress test

To test the integrated OPC UA server in the B&R PLC, a stress test of the system was performed. The goal of this experiment was to find out, how the RTT of the ReadSafetyData method changes, depending on the server workload. For this, the computer as shown in Figure 4.11 created threads with OPC UA clients. These clients would send 1000 attribute read requests per second each to the server. The tests ranged from one to ten threads and each one was run for approximately one minute.

With Wireshark [27], the network traffic created during this test was captured and stored in a .pcapng file. This file is then used in a python script to analyze the data (see Listing 6.3). To reduce the amount of data the script has to evaluate, the Wireshark file was filtered, so that only the ReadSafetyData method calls (requests and responses) to the PLC spammed with attribute reads are shown. The script then iterates over all ten files and calculates the average RTT of the method call. Furthermore, it stores the minimum and maximum values and calculates the 95% confidence interval. Figure 4.13 shows the results of this test.

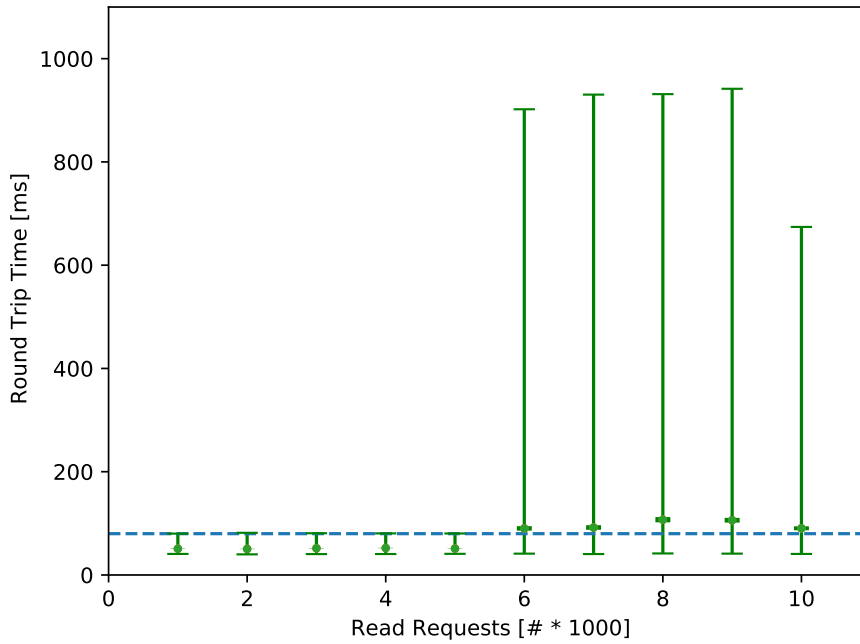


Figure 4.13: Stress test results

We can see that up until five threads the RTT lays within a much tighter interval than from six to ten threads. This can be attributed to the method timeout, which only occurs if more than five threads are used. This timeout plus the time it takes to reconnect to the server can increase the RTT to over 1s.

Figure 4.14 shows a cutout of Figure 4.13 to closer examine the results of the test. We can see, that for all tests, the minimum value is about 40ms. In the tests with more clients, these low values are captured at the beginning, where not as many requests are queried on the server. For one to five threads the max value is just about, or slightly over the timeout, meaning that it could be possible for the application to timeout even at a low workload. For six to ten threads the max value as seen before can be as high as 1s.

The average RTT, represented by the green dot, for one to five clients, lays at about 51ms. For six and more, the averages vary from 90ms to 100ms. The dark green bar represents the 95% confidence interval. For the first five tests, we can see, that this interval is very small, as opposed to the last five, where it is more spread apart. This interval and the average show that in general, the application will not timeout if five or fewer clients are used, but as soon as 6 or more clients send requests, the application changes into a safe state most of the time.

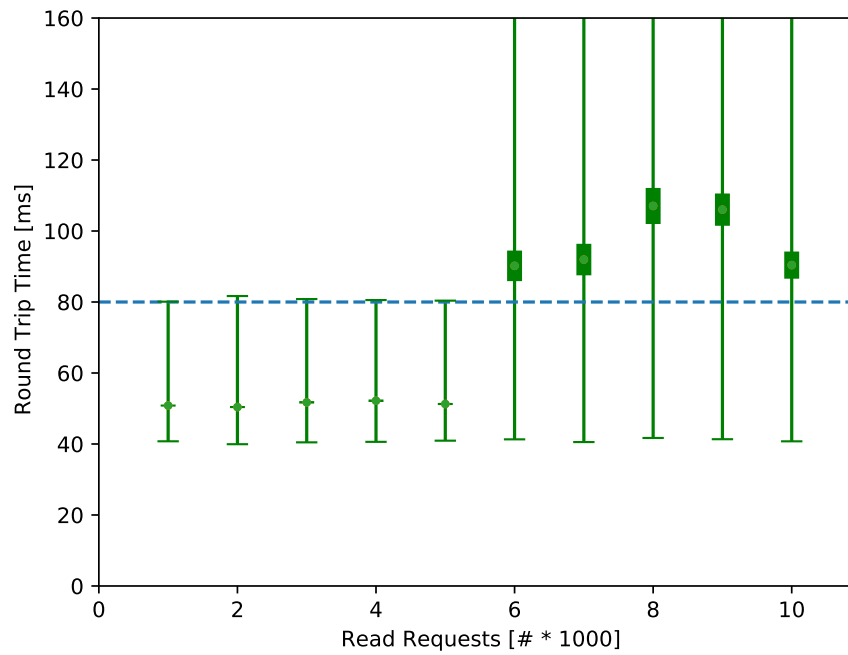


Figure 4.14: Cutout of stress test results

The stress test showed, that the estimated worst-case RTT was underestimated. Even at a low workload, some requests timed out, meaning that the application would change into a fail-safe state. To fix this behavior, the timeout set for the method needs to be increased to 100ms, which is still in a valid range for standard safety applications.

Conclusion and future work

Motivated by the publication of the specification *OPC UA Part 15: Safety* by the OPC Foundation, this thesis presented a simplified approach for safe communication between two PLCs. To achieve this, a closer look at the specification was provided and then some of the ideas were used to create a safe communication example.

The example consists of two B&R PLC systems, which exchange data via OPC UA. Each of these systems runs an application, which simulates the control of a saw system. To enable a bidirectional communication, each PLC hosts its own OPC UA server. To make it possible for a client to read safety data from the server, the information model from the specification is used and the `ReadSafetyData` method is implemented. The application then uses the safety data provided by the method response to decide whether or not to change to a fail-safe state.

The evaluation of this application showed that the communication between the two PLCs works well, but depends heavily on the cycle time of the PLC and on the timeout set for a method call. As soon as external systems start to send requests to the server, the functionality of the safe system could get interrupted. Depending on how many requests are sent from the external client, the application changes to a fail-safe state, if the server cannot respond in time.

The stress tests showed, that if more than five external clients sent 1000 read requests per second to the server, the RTT can get as big as 1s and the average is put over 80ms, meaning that the application will timeout most of the time. With less than five external clients, the application works, but could still timeout, as some RTTs are still over 80ms. To fix this behavior the timeout would need to be set to 100ms, which is still acceptable for safety applications. The results of this stress test show that the safety function works, but more importantly, it illustrates the need for the timing analysis of the whole system and the worst-case RTT.

Future work could make an extension to this application, by implementing more of the concepts described in the specification, such as CRC checks or monitoring number. Also, the diagnostic interface could be added to get a reading of the inner states of the application. Because of the work the OPC Foundation is currently putting into developing and describing the interfaces and the stack used, it might also be possible for future theses to incorporate these interfaces and the provided stack to create a complete safety system as described in the specification.

Appendix

6.1 Example Program

```

//check safety components for errors
bSFState_ok = diStateEStop && diStateLightCurtain && SafetyData_c.
    OutSafetyData;
bSFFub_ok = !comError_SF_EStop && !comError_SF_ESPE && !
    comError_SF_ModeSelector;

//set request to start saw
if(diStartStopQuit==true && diStartStopQuit_old!=diStartStopQuit &&
    comMode_Auto==true && bResetRequest==false)
    bCmdSaw = true;
else if(diStartStopQuit==true && comMode_Hand==true && bMode_Hand==true)
    bCmdSaw = true;

//set request to stop saw
if(diStartStopQuit==true && diStartStopQuit_old!=diStartStopQuit &&
    comMode_Auto==true && diStateSawOn==true)
    bCmdSaw = false;
else if(diStartStopQuit==false && comMode_Hand==true && diStateSawOn==true)
    bCmdSaw = false;

//safety components not ok, set request for error reset
if(bSFFub_ok==false || bSFState_ok==false){
    bCmdSaw = false;
    bResetRequest=true;
    bMode_Hand=false;
}
comReset_SF = false;

//reset errors
if(bResetRequest==true && diStartStopQuit==true && diStartStopQuit_old!=
    diStartStopQuit){

```

```
comReset_SF = true;
bResetRequest = false;
}
if(diStartStopQuit==false && diStartStopQuit_old!=diStartStopQuit)
    bMode_Hand=true;

//set saw output
doSaw = (bCmdSaw && bSFfub_ok && bSFState_ok);
diStartStopQuit_old=diStartStopQuit;
```

Listing 6.1: Example Programm in ANSI C

6.2 OPC UA method definition

```
OPCUA_METHOD ReadSafetyData
    ARG_INPUT
        InSafetyConsumerId : UInt32 := SafetyData.InSafetyConsumerId;
        InMonitoringNumber : UInt32 := SafetyData.InMonitoringNumber;
        InFlags : Byte := SafetyData.InFlags;
    END_ARG

    ARG_OUTPUT
        OutSafetyData : Boolean := SafetyData.OutSafetyData;
        OutFlags : Byte := SafetyData.OutFlags;
        OutSPDU_ID_1 : UInt32 := SafetyData.OutSPDU_ID_1;
        OutSPDU_ID_2 : UInt32 := SafetyData.OutSPDU_ID_2;
        OutSPDU_ID_3 : UInt32 := SafetyData.OutSPDU_ID_3;
        OutSafetyConsumerID : UInt32 := SafetyData.
            OutSafetyConsumerID;
        OutMonitoringNumber : UInt32 := SafetyData.
            OutMonitoringNumber;
        OutCRC : UInt32 := SafetyData.OutCRC;
        OutNonSafetyData : Boolean := SafetyData.OutNonSafetyData;
    END_ARG
END_OPCUA_METHOD
```

Listing 6.2: Method declaration of ReadSafetyData

6.3 Evaluation

```
#!/usr/bin/env python
from scapy.all import *
import scipy.stats
import numpy as np
import matplotlib.pyplot as plt

means = []
max_val = []
min_val = []
```



```

yerr = []
conf_int = []
directory = "./BuRLogs/"
fileList = [
    "1x1000_opc.pcapng",
    "2x1000_opc.pcapng",
    "3x1000_opc.pcapng",
    "4x1000_opc.pcapng",
    "5x1000_opc.pcapng",
    "6x1000_opc.pcapng",
    "7x1000_opc.pcapng",
    "8x1000_opc.pcapng",
    "9x1000_opc.pcapng",
    "10x1000_opc.pcapng"]

for filename in fileList:
    packets = rdpcap(directory + filename)
    idx = 1
    m = []
    for request in packets:
        if request[IP].src == '192.168.1.30' and request[IP].len < 200:
            for response in packets[idx:]:
                if np.array_equal(np.array(response.load[16:20]), np.array(
                    request.load[16:20])) == True:
                    m += [(response.time - request.time)*1000]
                    break

            idx += 1

    means += [np.mean(np.array(m))]
    max_val += [np.max(np.array(m))]
    min_val += [np.min(np.array(m))]
    conf_int += [scipy.stats.sem(np.array(m))*scipy.stats.t.ppf((1+0.95)/2.,
        len(m))]

yerr = [np.array(means)-np.array(min_val), np.array(max_val)-np.array(means)]

fig, ax = plt.subplots()
ax.plot(np.arange(0, 12, 1), np.full((12,1), 80), '--')
ax.errorbar(np.arange(1, 11), means, yerr=yerr, fmt='.', ecolor='g', capthick
    =1, capsize=5)
ax.errorbar(np.arange(1, 11), means, yerr=np.array(conf_int)/2, fmt='.',
    ecolor='g', elinewidth=7)
ax.set_ylabel('Round Trip Time [ms]')
ax.set_xlabel('Read Requests [# * 1000]')

plt.xlim(0, 11)
plt.ylim(0, 1100)
plt.savefig("results.pdf")

plt.ylim(0, 160)
plt.savefig("results_cut.pdf")

```

Listing 6.3: Evaluation script

List of Figures

2.1	The foundation of OPC UA [2]	4
2.2	OPC UA layered architecture [2]	4
2.3	OPC UA graphical notation (adapted from [2])	5
2.4	OPC UA simple model	5
2.5	OPC UA simple model	6
2.6	OPC UA extended model	7
3.1	Safety layers [11]	12
3.2	Server Objects for OPC UA Safety	13
3.3	Safety Parameters for SafetyProvider	14
3.4	State Diagram for SafetyProvider [11]	17
3.5	State Diagram for SafetyConsumer [11]	18
3.6	Relationship between standards [11]	19
4.1	System setup	22
4.2	Hardware setup of the Example Project [22]	23
4.3	Assignment of variables in the Safe View	23
4.4	Configuration of output module	24
4.5	Input Configuration for the Light Curtain	24
4.6	SafetyProvider model in UA Modeller	25
4.7	SafetyParameter model in UA Modeller	26
4.8	Reference for ReadSafetyData	27
4.9	States of the Server	27
4.10	States of the Client	28
4.11	Test setup	29
4.12	Sequence diagram of the communication	29
4.13	Stress test results	31
4.14	Cutout of stress test results	32

List of Tables

3.1	Signatur of ReadSafetyData	15
3.2	Signatur of ReadSafetyDiagnostics	15
3.3	Calculation of SPDU_ID	16
4.1	Components of the PLC	21
4.2	Supported Versions on B&R Systems	26

Listings

6.1	Example Programm in ANSI C	35
6.2	Method declaration of ReadSafetyData	36
6.3	Evaluation script	36

Bibliography

- [1] J. Pfrommer and T. Usländer, “Open-Source Implementierung von OPC UA PubSub für echtzeitfähige Kommunikation mit Time-Sensitive Networking,” in *Kommunikation und Bildverarbeitung in der Automation*, pp. 78–87, Springer, 2020.
- [2] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer Science & Business Media, 2009.
- [3] OPC Foundation, *OPC Unified Architecture Part 1: Overview and Concepts*, release 1.04 ed., 11 2017.
- [4] OPC Foundation, *OPC Unified Architecture Part 2: Security Model*, release 1.04 ed., 08 2018.
- [5] OPC Foundation, *OPC Unified Architecture Part 3: Address Space Model*, release 1.04 ed., 11 2017.
- [6] OPC Foundation, *OPC Unified Architecture Part 4: Services*, release 1.04 ed., 11 2017.
- [7] OPC Foundation, *OPC Unified Architecture Part 5: Information Model*, release 1.04 ed., 11 2017.
- [8] OPC Foundation, *OPC Unified Architecture Part 6: Mappings*, release 1.04 ed., 11 2017.
- [9] OPC Foundation, *OPC Unified Architecture Part 8: Data Access*, release 1.04 ed., 11 2017.
- [10] OPC Foundation, *OPC Unified Architecture Part 14: PubSub*, release 1.04 ed., 02 2018.
- [11] OPC Foundation and PROFIBUS Nutzerorganisation e.V., *OPC Unified Architecture Part 15: Safety*, release 1.00 ed., 10 2019.
- [12] P. Gehlen, *Funktionale Sicherheit von Maschinen und Anlagen: Umsetzung der Europäischen Maschinenrichtlinie in der Praxis*. Wiley, 2010.

- [13] J. Börcsök, *Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme*. VDE VERLAG, 2014.
- [14] IEC, “Functional safety of electrical/electronic/programmable electronic safety-related systems,” IEC 61508, 2010.
- [15] ISO, “Safety of machinery – safety-related parts of control systems – part 1: General principles for design,” ISO 13849-1, 2015.
- [16] IEC, “Functional safety - safety instrumented systems for the process industry sector,” IEC 61511, 2016.
- [17] IEC, “Industrial communication networks - profiles - part 3: Functional safety fieldbuses - general rules and profile definitions,” IEC 61784-3, 2017.
- [18] ISO, “Safety of machinery – safety-related parts of control systems – part 2: Validation,” ISO 13849-2, 2012.
- [19] ISO, “Safety of machinery - general principles for design - risk assessment and risk reduction,” ISO 12100, 2010.
- [20] IEC, “Safety of machinery - functional safety of safety-related electrical, electronic and programmable electronic control systems,” IEC 62061, 2005.
- [21] IEC, “Electromagnetic compatibility (emc) - part 6-7: Generic standards - immunity requirements for equipment intended to perform functions in a safety-related system (functional safety) in industrial locations,” IEC 61000-6-7, 2014.
- [22] “Arbeiten mit dem SafeDESIGNER.” <https://www.br-automation.com/de-at/academy/seminare-in-oesterreich/trainingsmodule/sicherheitstechnik/tm510-arbeiten-mit-dem-safedesigner/>. Accessed: 2017-05-22.
- [23] “Automation Studio 4.7.” <https://www.br-automation.com/de-at/downloads/>. Accessed: 2020-03-01.
- [24] “Automation Studio SafeDESIGNER 4.4.” <https://www.br-automation.com/de-at/downloads/>. Accessed: 2020-03-01.
- [25] “Unified Automation - UA Modeler.” <https://www.unified-automation.com/products/development-tools/uamodeler.html>. Accessed: 2020-04-01.
- [26] “Unified Automation - UA Expert.” <https://www.unified-automation.com/products/development-tools/uaexpert.html>. Accessed: 2020-04-01.
- [27] “Wireshark.” <https://www.wireshark.org/>. Accessed: 2020-03-01.