Master's Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Portable Application-Specific Web Server for Devices with OPC UA

Bc. Jiří Kalousek, MSc.

Supervisor: Ing. Pavel Burget, Ph.D.
Field of study: Cybernetics and Robotics
May 2019

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kalousek Jiří** Personal ID number: **420196**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Portable application-specific web server for devices with OPC UA

Master’s thesis title in Czech:

Přenositelný webový server pro specifické aplikace na zařízeních s OPC UA

Guidelines:

In the previous series of safety-critical railway devices developed at Siemens CT Prague a web server and a web client were used which provided view of system diagnostics. In the next version of devices the web server is replaced by an OPC UA Server.
1. Study the topic of diagnostics of distributed industrial systems.
2. Explore the origitnal solution of the web server and client providing view on the system diagnostics of the FM platform (safety-critical embedded platform)
3. Explore the way how diagnostics is solved using OPC UA.
4. Design and implement the connection of the original user interface with the OPC UA. Implement the server and a corresponding application-specific client.

Bibliography / sources:

[1] A. Braune, S. Hennig, and S. Hegler, ‘Evaluation of OPC UA secure communication in web browser applications’, in 2008 6th IEEE International Conference on Industrial Informatics, Daejeon, South Korea, 2008, pp. 1660–1665.
[2] S. X. Ding, Model-based fault diagnosis techniques: design schemes, algorithms and tools, 2nd ed. New York: Springer, 2013.
[3] S.-H. Leitner and W. Mahnke, ‘OPC UA - Service-oriented Architecture for Industrial Applications’, Softwaretechnik-Trends, vol. 26, no. 4, 2006.
[4] OPC Foundation, ‘OPC Unified Architecture Specification’, Unified Architecture. [Online].

Name and workplace of master’s thesis supervisor:

Ing. Pavel Burget, Ph.D., Testbed, CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **14.02.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

Ing. Pavel Burget, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt

Student’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I would first like to thank the experts from Siemens who enabled me to work on a real project which has a further utilization. In particular, I am grateful to Ing. Charles Bajeux for his humor, patience, and support in Javascript.

I would also like to thank my friend Alain Perkaz, MSc. for his valuable comments on this work. Our cooperation on related semestral projects in Sweden gave me a great lesson.

Next, I want to thank my supervisor, Ing. Pavel Burget, Ph.D. who took charge of guiding my thesis and helped me with organization matters.

Finally, I must express my gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study.

Declaration

I hereby declare that I worked on the presented thesis independently and I cited all references in accordance with the Methodical Instruction no. 1/2009 about ethical principles for academic thesis writing.

In Prague, 24th May 2019

Abstract

Communication plays a vital role in the nowadays technological world, and information poses a considerable value. The automation in industry is increasingly widespread. From sensors and actuators up to workstations with enterprise and diagnostic applications, all devices have to be connected to the network, and they have to communicate among themselves. The interconnection can be achieved thanks to the standardization effort which resulted, among others, in the OPC UA protocol. This thesis is concerned about modern ways of industrial communication, especially about retrieving diagnostics data. It is aimed to design an interconnection between an existing web-based Human-Machine Interface and an OPC UA server introduced into a new version of safety-critical railway controllers developed by Siemens Mobility, s.r.o. in Prague. The result is a software solution which can obtain data from an OPC UA server, perform mapping of semantics, convert data formats, and provide the data to the original web-based interface. The proposed concept can be applied when interconnection between customized user interfaces and unified diagnostic servers needs to be achieved.

Keywords: OPC UA standard, OPC UA client, web server, diagnostics, industrial communication, Human-Machine Interface

Supervisor: Ing. Pavel Burget, Ph.D.

Abstrakt

Komunikace hraje v dnešním světě plném technologií klíčovou roli a informace představují nemalou hodnotu. Do průmyslového prostředí je na všech úrovních čím dál více zaváděna automatizace. Zařízení od senzorů a akčních členů až po pracovní stanice poskytující přístup k plánování a diagnostice systémů musí být připojená k síti a vzájemně si vyměňovat data. K tomu významně dopomohla standardizace komunikační platformy v podobě OPC UA protokolu. Tato práce se zabývá moderními způsoby komunikace v průmyslových sítích, zejména pak získáváním diagnostických dat. Cílem práce je návrh propojení existujícího webového rozhraní mezi člověkem a strojem s OPC UA serverem zavedeným do nové řady bezpečnostně-kritických kontrolérů pro železniční zabezpečovací zařízení vyvíjené v Siemens Mobility, s.r.o. v Praze. Výsledkem je softwarové řešení, které získává data z OPC UA serveru, provádí mapování odlišných sémantik, převádí data do požadovaného formátu a poskytuje je původní webové aplikaci. Navržený koncept může být využit při propojování specifických uživatelských rozhraní s unifikovanými diagnostickými servery.

Klíčová slova: OPC UA standard, OPC UA klient, webový server, diagnostika, průmyslová komunikace, rozhraní mezi člověkem a strojem

Překlad názvu: Přenositelný webový server pro specifické aplikace na zařízeních s OPC UA

Contents

1 Introduction 1

1.1 Problem Outline 2

1.2 Thesis Structure 3

2 Background 5

2.1 Evolution of Fieldbus Standards . 5

2.2 Architectures of Industrial Networks 8

2.3 Railway Signaling Systems 11

2.4 The OPC UA Standard 15

3 Existing OPC UA Tools and Implementations 25

3.1 OPC UA Testing Servers 25

3.2 OPC UA Web-based Clients . . . 27

3.3 OPC UA Software Libraries and Tools 28

4 The Legacy Software Architecture 33

4.1 User Interface 34

4.2 Metadata 36

5 Design and Implementation 37

5.1 Motivations and Design Decisions 37

5.2 Requirements Analysis 39

5.3 OPC UA Test Server 41

5.4 Metadata Mapping 42

5.5 Software Design 45

5.6 Implementation 48

5.7 Library fix 49

5.8 Deployment 50

5.9 Validation 51

6 Conclusion 53

A Bibliography 55

B Example of the Web Server REST API 59

C Example JSON Object for Metadata Mapping (Lookup Table) 61

D UML Deployment Diagram 63

E UML Class Diagram 65

Figures

2.1 Milestones of fieldbus evolution [1]. 6

2.2 Example of Fieldbus integration in PROFInet (Industrial Ethernet) [2]. 7

2.3 Software architecture of a SCADA system [3]. 8

2.4 Three-tier architecture pattern [4]. 9

2.5 State-of-the-art distributed control system [3]. 10

2.6 Comparison of centralized and decentralized signaling control architecture in railway [5]. 12

2.7 Demonstration of the Siemens project which employs decentralized signaling control architecture. The colored squares represent controllers of various field elements along the railroads [6]. 13

2.8 The state-of-the-art reference architecture for railway signaling systems designed by Eulynx initiative[7]. 15

2.9 OPC UA Node Model [8]. 16

2.10 OPC UA Object Model [8]. . . 19

2.11 The data model of a DataVariable with history [8]. 20

2.12 The OPC UA stack overview [8]. 22

3.1 Graphical user interface of Prosys OPC UA Simulation Server. 26

3.2 Graphical user interface of OPC UA Server Simulator from Integration Objects. 26

3.3 Architecture of Prosys OPC UA Web Client [9]. 28

3.4 OPC UA Web Based Client developed by One-Way Automation. 28

3.5 Illustration of the Free OPC UA Modeler tool [10]. 31

4.1 Block diagram of the legacy architecture. 34

4.2 The user interface of Signal Control Module (SCM) as an example of a Web Client Application that was supposed to use the OWA for interconnection with a target OPC UA server. 35

4.3 Meta model of the Client Schema (structure of XML content). Capital letters are used in literal names of XML elements. 36

5.1 Prospective deployment of the OPC UA Website Adapter. 39

5.2 Example of how the test server is modeled according to the original metadata structure. Note that only *HasComponent* and *Organizes* relationships are displayed. In reality, there are many other relationships among the nodes. For example, the node *DiagnosticRoot* has a relationship *HasTypeDefinition* with the node *CSFolderType*. 42

5.3 Mapping diagram showing the metadata mapping problem. 44

5.4 UML Use Case diagram for the OWA application. 45

5.5 UML Component Diagram for the OWA application. 46

5.6 UML Sequence Diagram for the OWA application. 47

5.7 Visualization of the queue that stores changed property values in time. Each set of property values at a given point in time is referred to as objects. 49

Tables

2.1 Overview of the Communication Profile Families according to IEC 61784. [11] [1] 6

2.2 General protocol stack for safe railway communication [12]. 13

2.3 Attributes of the Base NodeClass. Each node has these essential attributes. Derived NodeClasses add more class specific attributes.[8]. . . 17

2.4 NodeClasses derived from the Base NodeClass in OPC UA. [8]. 18

2.5 Overview of Service Sets. 21

3.1 Open source OPC UA client and server implementations [13] available for commercial use. 29

4.1 Initial objects requested by the web page. 35

5.1 Functional requirements 40

5.2 Non-functional requirements . . . 41

5.3 Mapping of data types between OPC UA and the proprietary Client Schema format. 43

5.4 Validation of functional requirements 52

Introduction

From the invention of the steam engine and machine manufacturing at the end of 18th century, through the rise of mass production in the late 19th century, and the introduction of automation in the 1970s, our civilization is getting closer to the next revolution, so-called 4th Industrial Revolution. It combines technologies from many areas and builds upon the 3rd Industrial Revolution that enabled automation thanks to electronics and the information technology. It blurs the boundaries between the physical and digital world in so-called cyber-physical systems. Advancements in areas like robotics, artificial intelligence, autonomous vehicles, or biotechnology will influence billions of people around the globe [14]. The trend in the interconnection of people, data, things, and processes is going towards the future Internet of Everything (IoE) - "a network of networks where billions or even trillions of connections create unprecedented opportunities as well as new risks" [15].

In industry, the traditional production systems are getting transformed into networked manufacturing systems that are networked in the so-called Industrial Internet of Things (IIoT). Such manufacturing systems are formed by autonomous robotic machines equipped with many sensors. Manufactured products are composed of smart parts that contain information about themselves. Thanks to this, nearly every product can be customized [16]. Communication technology and interoperability play a vital role since every participant is strongly dependent on the interconnection with others. Very important is also the process of standardization and unification because all the interconnected technology needs a common platform for mutual communication.

Various Ethernet-based protocols were supposed to unify the communication in the first decade of the new century, but the effort was only partially successful [2]. There are protocols like Profibus, Interbus, CANopen, etc. that needed a common platform for data exchange. Other protocols like Profinet or PowerLink have been developed to integrate the existing ones. Finally, companies have signed cooperation agreements with the OPC Foundation [17] which resulted in the establishment of the *OPC UA protocol*. The original field level communication space was preserved. It was unified in vertical and horizontal levels from sensors and actuators to enterprise management systems by the OPC UA protocol. Information can now be shared between previously incompatible systems, and industrial communication can further expand [2]. The evolution also implies the importance of service-oriented architectures which allow easier development and deployment of decentralized and interoperable systems.

1.1 Problem Outline

In this thesis, the OPC UA protocol is further elaborated regarding a practical application in diagnostics of a new version of safety-critical railway controllers developed at Siemens Mobility, s.r.o in Prague. The functional objective of this work was to create a software adapter that would link up an original existing web-based user interface with an OPC UA server introduced into a new version of the controllers. The provisional title for this project was *OPC UA Website Adapter* (hereinafter OWA). The software developed within this thesis is also referred to as the *OWA application*.

The existing user interface is a frontend application implemented in HTML, CSS, and Javascript. The web page is written as a single-page application [18] that is composed of a few functional modules. The modular architecture is similar to projects designed with Vue.js framework [19] for developing user interfaces. One of the modules is responsible for communication with a corresponding web server using HTTP requests and WebSockets. For this thesis, the interface between a web client and the original web server needed to be clearly described in preparation for the ensuing software design.

The OPC UA server, which was supposed to reside at the other end of the OWA, had not been designed yet by the time of working on this thesis. Thus, an appropriate OPC UA test server had to be prepared. The OWA interconnecting the frontend application with the OPC UA server had to be designed in a way so that it could be easily adjusted to a different target server structure. It also had to be capable of metadata mapping between an

arbitrary target OPC UA server and the existing user interface connected as a web client. The mapping had to be designed in a scalable way since the details about potential target OPC UA servers were not known. The server part of the OWA had to provide the same API as the original server in the previous version of the diagnostic system.

1.2 Thesis Structure

This thesis is organized as follows. Chapter 2 elaborates the theoretical knowledge at a high level. Therefore, the theory is not explained in too much detail since this chapter should serve as an introduction to the technology related to the solved problem. It includes industrial communication networks, platform architectures, railway signaling systems, and details about the OPC UA protocol. Chapter 3 discusses some of the existing tools and implementations regarding the OPC UA technology. Chapter 4 analyzes the current solution which was supposed to be reused. Based on the reviewed theory and existing solutions, Chapter 5 presents motivations for the intended work as well as the consequent design and implementation of the developed software solution. Finally, Chapter 6 concludes the project, summarizes contributions and mentions future work.

Chapter 2

Background

2.1 Evolution of Fieldbus Standards

Since the early 1970s, many industrial computer network protocols have been developed to help with control of distributed industrial automated systems. Such a computer network interconnects Programmable Logic Controllers (PLC) with sensors and actuators that do the actual work in manufacturing. This kind of low-level network is generally referred to as *Fieldbus* network. MIL-STD-1553 Fieldbus protocol is regarded as the first real Fieldbus. It has a serial transmission of control and data information, master-slave structure, the possibility to cover longer distances, and it has integrated controllers [1]. The Figure 2.1 shows the main milestones in the evolution of Fieldbus protocols. Some of them extended already existing ones.

In the mid-1980s, the standardization of Fieldbus protocols started, but it was taking place mainly on a national level. The international standardization was not successful until the end of the 1990s when several national standards were accepted as European (EN) standards. These standards were then grouped in an international standard IEC 61158. Unfortunately, the interoperability between particular Fieldbus protocols was not ensured because different protocols were suitable for various applications. The IEC 61158 grouped eight different Fieldbus protocols (Type 1 to Type 8), and a guideline was provided to show how different parts can be compiled in a functioning system [20]. Later in 2008, many additional Fieldbus protocols were added, and all of them were reorganized into Communication Profile Families (CPF) in a new IEC 61784 standard (Table 2.1).

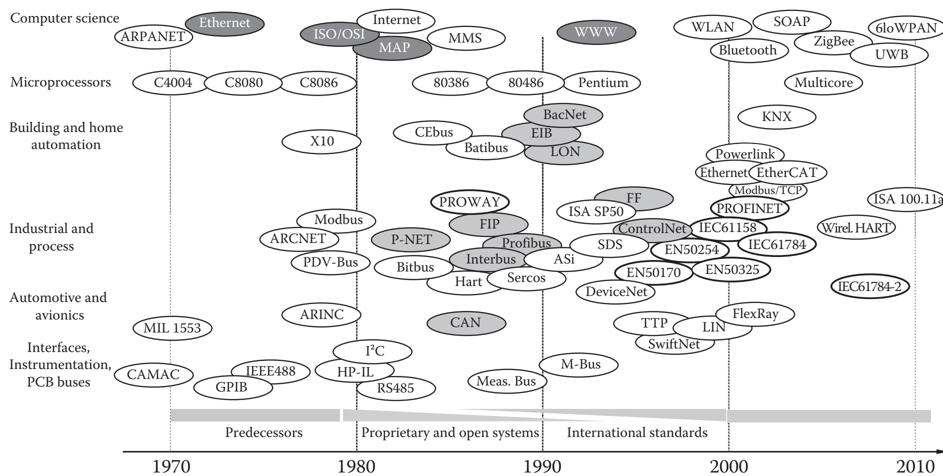


Figure 2.1: Milestones of fieldbus evolution [1].

Profile	Brand Name
CPF-1	Foundation Fieldbus (H1, HSE, H2)
CPF-2	ControlNet, EtherNet/IP
CPF-3	Profibus-DP, Profibus-PA, PROFInet
CPF-4	P-Net RS-485, P-Net RS-232
CPF-5	WorldFIP (MPS, MCS, SubMMS)
CPF-6	Interbus, Interbus TCP/IP, Interbus subset
CPF-7	Swiftnet transport, Swiftnet full stack
CPF-8	CC-Link
CPF-9	HART
CPF-10	Vnet/IP
CPF-11	TCnet
CPF-12	EtherCat
CPF-13	Ethernet Powerlink
CPF-14	EPA
CPF-15	MODBUS-RTPS, MODBUS TCP
CPF-16	SERCOS

Table 2.1: Overview of the Communication Profile Families according to IEC 61784. [11] [1]

The evolution of Fieldbus protocols continued towards the *Industrial Ethernet*. Fieldbus protocols from different families were still not compatible, but with proxy devices, they could achieve some intercommunication. For

example, PROFInet has integrated several Fieldbus protocols as shown in Figure 2.2. The term Industrial Ethernet is used for protocols like PROFInet which use the Ethernet standard at the physical and data link layer. They usually also use IP protocol at the network layer and TCP/UDP at the transport layer of the ISO/OSI model.

Ethernet used to be considered inappropriate because of its lack of determinism and real-time capabilities. It was mainly implemented in Local Area Networks (LANs) at the office level. However, when switched Ethernet is used, and certain modifications are introduced, it can be utilized in the real-time domain instead of a traditional Fieldbus network. Modern Fieldbus protocols are still referred to as *Fieldbuses*, but they mostly make use of Ethernet on the lower levels. Therefore, nowadays, the boundary between LANs and Fieldbus networks is faded.

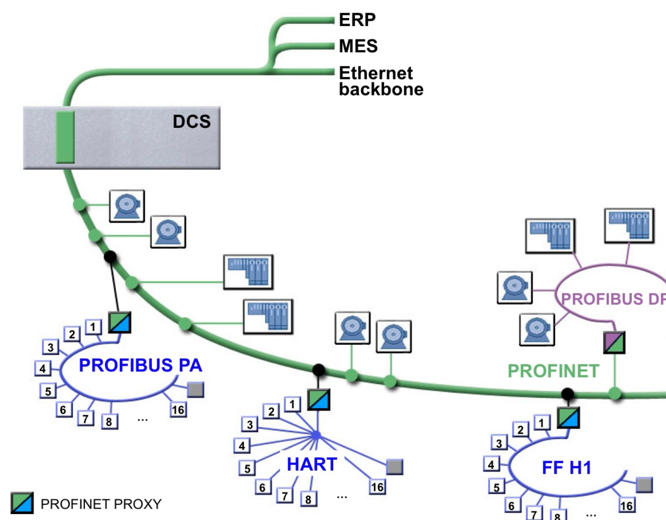


Figure 2.2: Example of Fieldbus integration in PROFInet (Industrial Ethernet) [2].

The modern Fieldbus protocols from the category of Industrial Ethernet use proprietary mechanisms on higher layers to meet specific needs. In this respect, they differ in a similar way like the *old* Fieldbus protocols did. Direct interoperability between industrial Ethernet solutions on lower levels is still not possible. Communication must be established on higher levels like it is done in PROFInet, or with the use of middleware layers like OPC [1].

In the future, the evolution of Fieldbus protocols will face the challenges of moving to the wireless domain. It may require even a complete redefinition of the lower Fieldbus protocol layers. The evolution also continues in safety-relevant systems. Various Fieldbus protocols add additional layers that aim to fulfill demanding safety requirements [1].

2.2 Architectures of Industrial Networks

The primary purpose of networks that are present at all levels of the manufacturing hierarchy is *Supervisory Control and Data Acquisition* (SCADA) systems. The issue is a software solution which coordinates diagnostics of machines and processes and allows their control from higher hierarchy levels. Diagnostics refers to monitoring the health and state of a tool, process, or system. Diagnostics data is transferred to the management layer and *Human-Machine Interface* (HMI). Control data is headed in the opposite direction as shown in Figure 2.3.

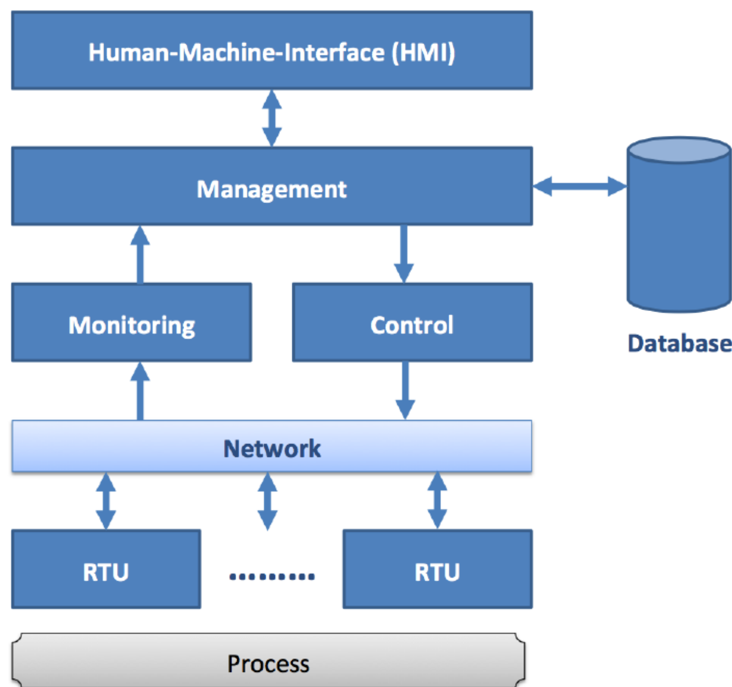


Figure 2.3: Software architecture of a SCADA system [3].

However, monitoring and control happen also at lower levels which interact with a manufacturing process where sensors and actuators are used. It is clear that diagnostics and control domains have diverse needs. Diagnostics require to transfer large amounts of data, and it usually needs to deal with data compression. Whereas control communicates lower volumes of data in shorter distances, but response time and determinism are critical. The authors in [21] divide computer networks in manufacturing into three domains - diagnostics, control, and safety. The safety domain has even more strict requirements for reliability and timing which implies that the realization of such a network can be quite challenging.

The continuing trend is to use the Ethernet protocol at all levels of manufacturing and to move towards consolidation of network standards. Here the traditional Ethernet-based network is advantageous for the collection of diagnostics data. However, standard Ethernet (IEEE 802.3) is not deterministic protocol, and it cannot guarantee network Quality of Service (QoS) that is crucial for low-level control. It has to be modified in terms of Media Access Control (MAC), packet encoding, and bandwidth optimization [21]. That is typical for Ethernet-based Fieldbus protocols.

Regarding the IIoT, there is a good comparison between IoT and IIoT in [22]. IoT specializes more on the design of new communication standards for new devices to be connected to the network. On the contrary, IIoT focuses on integration and interconnection of existing technology used in plants. "IIoT can be considered more an evolution rather than a revolution."[22] IIoT is a subset of IoT with specific requirements and applications. It aims at getting knowledge about the physical system, and it is not related to control applications at the field level. IIoT applications are usually concerned about supervision, optimization, and prediction. To do so, they have to collect a lot of diagnostic data.

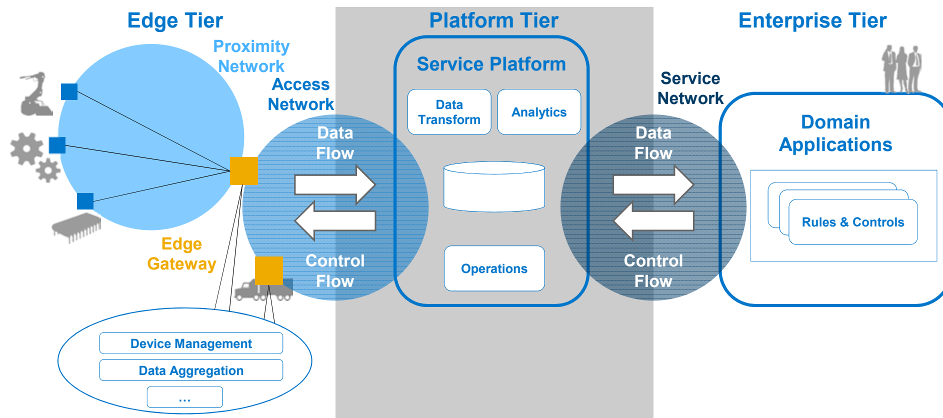


Figure 2.4: Three-tier architecture pattern [4].

There exist several reference architectures for IIoT as higher level abstractions that help with the design of different applications. It is usually a multilayer description of services provided at each level according to a particular application needs. Some architectures are reviewed in [4] by the IoT 2020 project team in the IEC Market Strategy Board. For example, the *Reference Architecture Model Industrie 4.0* (RAMI 4.0) specializes in industrial manufacturing systems of the next generation. Currently, the widely accepted architectural pattern has three tiers - edge, platform, and enterprise. These tiers are connected by proximity, access, and service networks. Figure 2.4

shows the arrangements for this pattern. The edge tier consists of sensors, actuators, and controllers interconnected by, so-called, proximity networks, which usually make use of Fieldbus protocols. The edge tier is connected to the platform tier via access networks and edge gateways. The platform tier is responsible for data processing and forwarding to the enterprise tier as well as relaying control commands down to the edge tier. The enterprise tier provides user interfaces and domain-specific applications.

The present state-of-the-art model of a distributed control system according to [3] and essentially also according to [4][22] is shown in Figure 2.4 and 2.5. The model in Figure 2.5 loosely follows the *Three-tier architecture pattern*. It consists of nodes placed in particular tiers interconnected with different networks. At the lowest-level, field devices, controllers, and gateways are interconnected via Fieldbus networks that can be redundant and they guarantee the deterministic behavior. Diagnostics data is usually transferred in free time slots and has lower priority than control data. The higher level uses Ethernet in a redundant configuration which exploits proprietary protocols that can take care of the network utilization. This level mediates the communication, handles data transformation, aggregation, etc. between controllers, gateways, servers, engineering tools, and user interfaces (HMI) from the enterprise level. Servers and HMIs can also be redundant. Behind the firewall, more domain applications may reside.

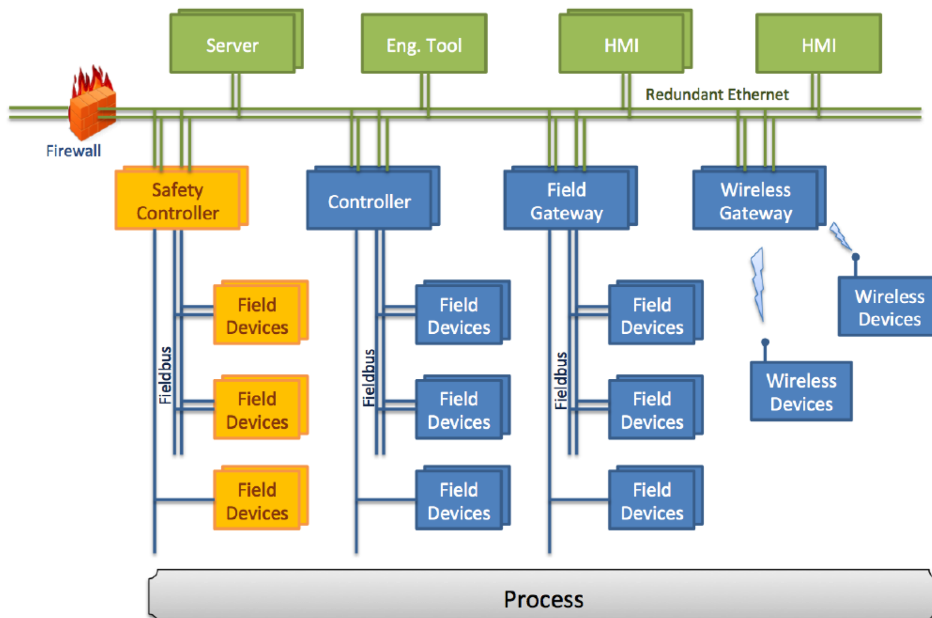


Figure 2.5: State-of-the-art distributed control system [3].

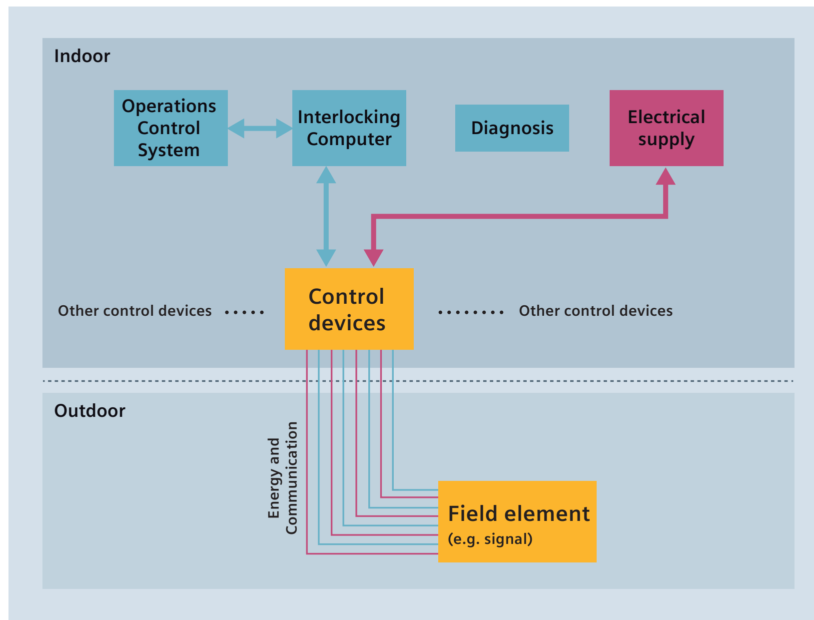
The interfaces between servers, engineering tools, user interfaces, and other nodes are usually implemented using the old OPC standard or the new OPC UA standard. The latter improves the interoperability among devices from different vendors and introduces platform independence.

2.3 Railway Signaling Systems

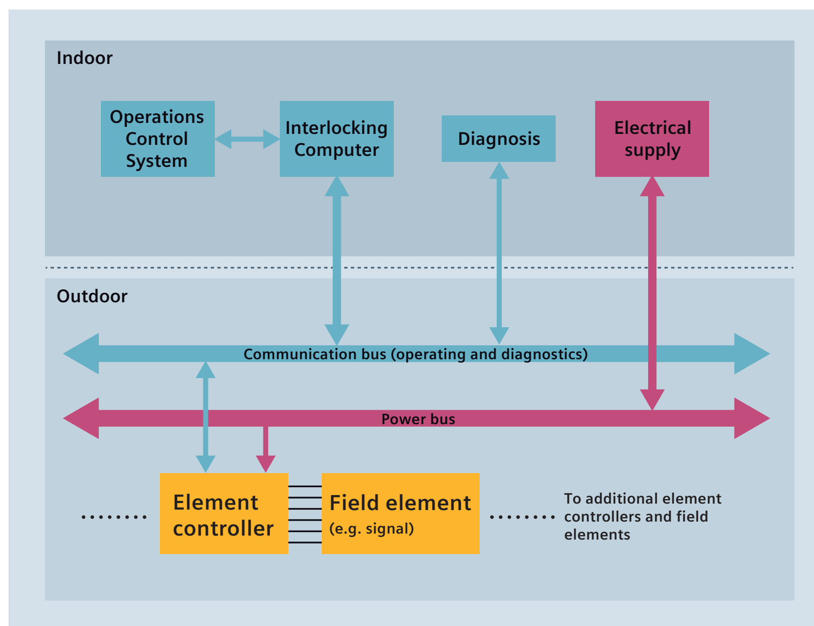
Although the evolution of Fieldbus systems has come a long way, the recent state-of-the-art network topology in the railway was still based on star formation of field elements and the control was done centrally from controllers placed indoor (Figure 2.6a). Finally, during the last years, decentralized control of elements has been introduced (Figure 2.6b). In the new organization, power and communication buses are separated. The *Element Controllers* (EC) that control level crossing systems, railroad switches, axle counters, etc. are placed outdoor, and delivery of the control data must be ensured in real-time via a communication bus. Ethernet protocol and optical fiber cables are used for longer distances. SHDSL[1] field rings and standard copper telephone lines maintain local connections within a station [5]. Siemens launched the first project which uses this model by the end of 2013. Figure 2.7 demonstrates the network organization in a real-life scenario. The communication in such a network is IP-based, but additional protocols must be implemented on top of the existing ones to assure safety.

Like there are different Fieldbus networks deployed in manufacturing, there are also different signaling systems used in the railway industry which are not interoperable. Deployment of different standards usually depends on countries or technology suppliers. The evolution in railway signaling standards loosely follows the evolution of Fieldbus protocols. Many different and incompatible signaling systems have been developed and deployed in European countries. In 2005, the European Commission and representatives from the rail industry signed a memorandum on the deployment of the *European Rail Traffic Management System* (ERTMS). Since then, the signaling systems in Europe are getting unified which brings interoperability on the European level. It should also reduce costs and improve efficiency and safety [23]. Currently, there exist several European initiatives like *SmartRail 4.0* or *Eulynx* that strive to standardize the system architecture and interfaces of railway signaling equipment.

[1] Single-pair high-speed digital subscriber line is a communication technology standard for symmetric data transfer over copper telephone lines.



(a) : The old structure of an electronic signaling control system (star formation).



(b) : The new structure of an integrated, decentralized signaling control architecture from Siemens.

Figure 2.6: Comparison of centralized and decentralized signaling control architecture in railway [5].

Generally, the communication protocol stack for railway applications can be designed as proposed by SmartRail 4.0 collaboration. It is shown in Table 2.2. It follows existing international standards including those related to safety-critical systems.

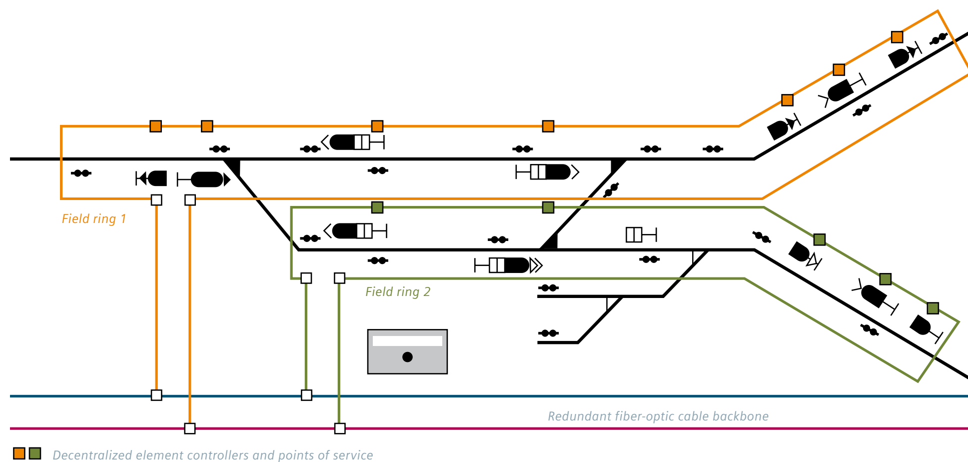


Figure 2.7: Demonstration of the Siemens project which employs decentralized signaling control architecture. The colored squares represent controllers of various field elements along the railroads [6].

Layer	Description
Transfer Contract	This layer creates transfer contracts which are data and protocol specifications. An application can define which protocols have to be used.
Transfer	This layer realizes a simple message-based protocol. It supports redundant application instances - message authentication code and addressing are supported.
Safety and Retransmission	This layer shall ensure a correct and safe end to end data exchange under SIL classification. Examples are RaSTA, EuroRadio, or RBC-RBC. It can be disabled in some cases.
Redundancy	This layer maintains sending data over redundant network interfaces and subsequent networks (e.g., Ethernet and GSM-R). It can be disabled in some cases.
Security	The communication has to be protected against unauthorized access (e.g., TLS).
Transport	Data transfer between endpoints is maintained by standard UDP/TCP protocol.
Network	Routing and addressing are maintained by the standard IP protocol.
Physical/Data Link	Physical and logical communication are maintained by protocols like Ethernet or GSM-R.

Table 2.2: General protocol stack for safe railway communication [12].

Like the Fieldbus protocols that are referred to as Industrial Ethernet protocols, the railway industry has its protocol based on Ethernet - *Rail Safe Transport Application* (RaSTA). The issue is a safety-critical network protocol which is an important step in the transformation of railway interlocking systems from proprietary blocks towards standardized and modularized systems. Its main goal is to ensure a fast and reliable connection between two communication endpoints in the safety-critical network. It is compliant with requirements for railway applications according to EN 50159 standard. RaSTA comprises of the *Safety and Retransmission Layer* and the *Redundancy Layer*. The Safety and Tetransmission Layer takes care of requesting retransmission for lost or altered packets. It also assures authenticity and integrity. The Redundancy Layer handles the correct sequencing of messages that are sent redundantly over distinct channels. It assures timeliness and availability. RaSTA uses 8 B checksum called *Safety Code* to ensure integrity. Data header and payload are hashed taking into account an initialization vector (so-called Network Key). The resulting Safety Code can be created only by verified entities within the network [24].

The endeavor to unify the railway signaling world is empowered, among others, by Eulynx. It is a European initiative linking interlocking subsystems that consists of 12 members - infrastructure managers of European countries. This initiative works on standardization of the technical architecture and the design methods in railway signaling. The job is divided into several Cluster Projects that focus on certain problems and domains. For example, the Modelling and Testing Cluster form rules for the system engineering process which has to be applied by each Cluster Project. Overall, the project management standardization is an important part of Eulynx activities. The *Eulynx Cluster Projects* are Reference Architecture, Assurance, Modeling and Testing, Data Preparation, Interface Interlocking - Radio Block Center, Interface Interlocking - Control System, Interface Interlocking - Train Detection System, Interface Interlocking - Light Signal, and Interface Interlocking - Level Crossing [7].

The *Reference Architecture* project forms a concept that is based on modern communication architectures and system design processes used in automation and telecommunication industry. It exploits IP-based communication using safe and secured closed and open networks [7]. Figure 2.8 visually demonstrates the architecture.

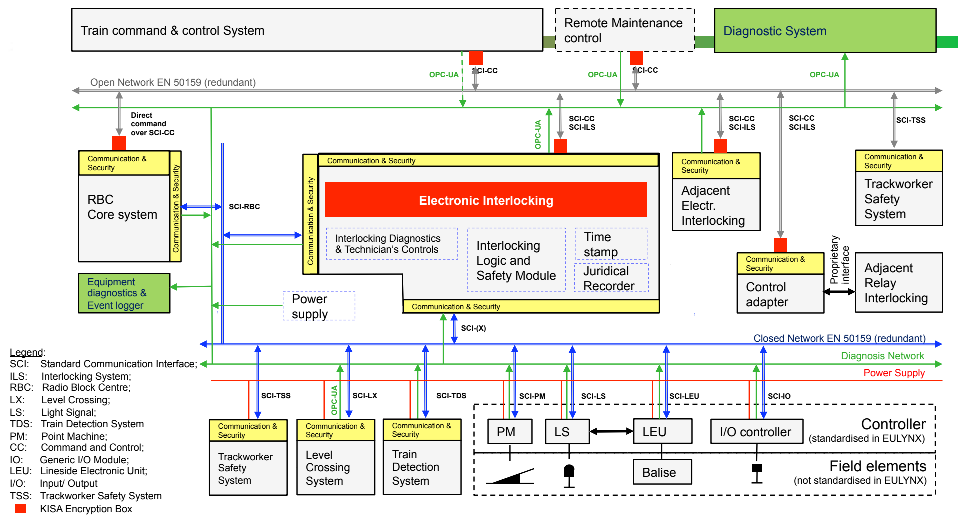


Figure 2.8: The state-of-the-art reference architecture for railway signaling systems designed by Eulynx initiative[7].

2.4 The OPC UA Standard

Before the specification of the *OPC Unified Architecture* (hereinafter OPC UA) was created, the industrial automation had been mostly using Microsoft's Component Object Model standard to ensure communication between particular object-oriented software components. The distributed version of this standard brought the system of Distributed Computing Environment and Remote Procedure Calls (RPC). Several automation vendors formed a new standard for data access called *Object Linking and Embedding* (OLE) *for Process Control* (OPC). Later, the OPC Foundation was established and defined standards for open connectivity of industrial automation devices [17] - specifications for Data Access, Alarm and Events, Historical Data Access, etc.

The OPC UA standard is the next step in the evolution of industrial standards. It is based on the original OPC, but it is platform independent, service-oriented, and it has improved security and scalability [17]. OPC UA was originally focused on industry, but it has appeared to be an appropriate solution for interoperability and data exchange in other fields as well [2].

For information exchange, two approaches can be used - client-server and publisher-subscriber. The communication is technology independent (manufacturer, operating system, programming language) and data can be shared among various devices from small sensors, over PLCs, to cloud servers and any other connected devices. Moreover, OPC UA together with Time-Sensitive Networking (TSN) standard enables real-time Machine-To-Machine (M2M) communication, and it can even be used in safety-critical applications [2].

The *OPC UA specification* [8] consists of several parts (Part 1 to Part 13) which describe the information model structure, semantics, and interactions between particular applications and endpoints. Each OPC UA server has its address space constituted by various nodes that can be accessed from a client or a different server. Data stored in the structure can have different formats defined either by OPC, other standard organizations, or arbitrary vendors. The format description is called metadata. It needs to be accessed by clients in advance to know which data can be requested and what is its correct representation. Almost any node can have a reference to any other node anywhere in the address space or in the network. It means that even nodes from different servers in different networks can be interrelated.

Data Model

Every piece of information is a node with attributes (fundamental characteristics) that has references to other nodes. Only attributes can keep data values. Nodes can have the meaning of classes or data types as well as particular instances like objects, variables, methods, events, relationships, etc. The Figure 2.9 shows a node element. Every node has an attribute *NodeClass* which classifies nodes into respective classes. These classes define additional attributes and references according to a certain purpose of a corresponding node.

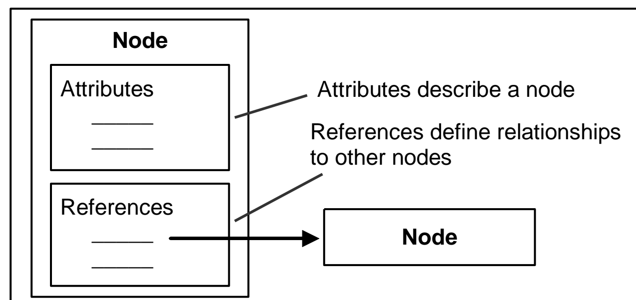


Figure 2.9: OPC UA Node Model [8].

Attribute	Description
NodeId	Unique identifier of a node. It must remain the same after rebooting of a server.
NodeClass	Identifies the NodeClass of a node.
BrowseName	This attribute consists of a namespace id and non-localized human-readable name of a node which should be used to browse a server in software (e.g., to get a specific child). It should not be displayed to a user.
DisplayName	This attribute should be used by clients to display the actual name of a node. It is localized which means the language of the name can be found out.
Description	Optional attribute explaining the meaning of a node.
WriteMask	This attribute states which attributes can be accessed by a client for write operations.
UserWriteMask	The same as WriteMask but it takes user access right into account.

Table 2.3: Attributes of the Base NodeClass. Each node has these essential attributes. Derived NodeClasses add more class specific attributes.[8].

The top parent NodeClass is called *Base NodeClass*. This class defines the basic attributes that each node must have (Table 2.3). It is the parent class for all other derived NodeClasses. Table 2.4 gives an overview of what kinds of nodes can be found in the OPC UA node structure, so-called *AddressSpace*.

Interconnected nodes physically constitute the AddressSpace and form the tree topology. All data is stored in leaves of this tree. It is in the attribute *Value* of nodes that are classified in *Variable NodeClass*. The difference between *Properties* and *DataVariables* is described in Table 2.4. OPC UA defines several built-in data types. Detailed explanation can be found in Part 6 of the OPC UA specification [8]. Additional data types can be defined as *DataType NodeClass* nodes with the use of built-in data types.

Object Model

The data structure is logically modeled by objects with variables and methods. Objects are nodes of *Object NodeClass* that behave like instances of object types (nodes of *ObjectType NodeClass*). An instance is linked with its definition by *HasTypeDefinition Reference*. The same states for a variable as an instance and its type. Variables are linked with objects using *HasProperty Reference* in case of Property and *HasComponent Reference* in case of

NodeClass	Description
ReferenceType	A node of this class defines a type of reference (e.g., HasProperty, HasComponent). A reference is then an instance of a specific ReferenceType. A reference is a part of a node and has no NodeClass.
View	A node of this class acts as the root for a group of nodes. The entire address space can be divided into specific groups (views).
Object	Class of a node which is an object - a real-world object or system component. Objects have properties, variables, methods, type definitions, etc. All of these are linked with an object by references.
ObjectType	This class defines how a subsequent instance (object) should look like - attributes, references, properties. Objects are always based on ObjectTypes.
Variable	There are two kinds of variables - Properties and DataVariables. Properties characterize nodes. DataVariables form the content of objects. DataVariables can have Properties, but Properties cannot have any DataVariables nor Properties.
VariableType	Nodes of this class provide type definitions for variables.
Method	Class of nodes that define callable functions.
DataType	Nodes of this class describe the syntax of a variable value.

Table 2.4: NodeClasses derived from the Base NodeClass in OPC UA. [8].

DataVariable. OPC UA defines a lot of various object types, variable types, and reference types. Custom types can also be defined if needed. Figure 2.10 visually represents an object in the OPC UA object model.

Methods are nodes defined by *Method NodeClass* which specify callable functions. Methods can have properties like *InputArguments* and *OutputArguments*. They get invoked using *Call Service*.

When a new empty OPC UA server is started, it is already populated with standardized nodes according to the OPC UA specification [8](details in Part 5 - Information Model). These nodes should be used by any custom architecture as much as it is possible and applicable.

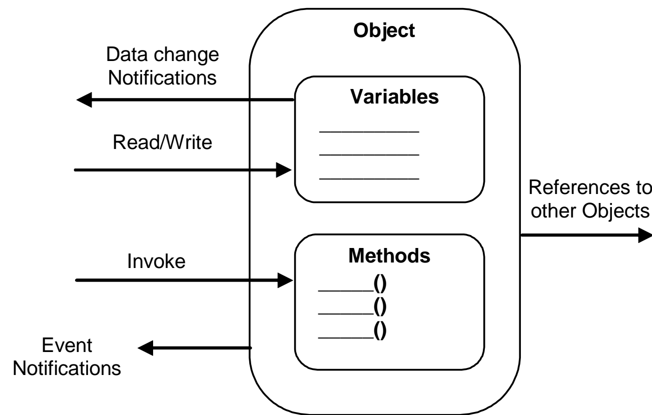


Figure 2.10: OPC UA Object Model [8].

Historical Data Access

For the access of historical data, there are special nodes (*HistoricalNode*, *HistoricalDataNode*, and *HistoricalEventNode*) where a client can request data from the past. Each variable has the attribute *Historizing* which indicates whether the server is collecting data for the history. A variable which history is backed up has an *Annotation* property and a reference *HasHistoricalConfiguration* to its *HistoricalAccessConfiguration* object. Figure 2.11 shows an example of how the history model can look like for a variable.

The Annotation property is metadata related to an item at a given instance in time. The actual data values are called *BoundingValues* and they are associated with a starting and ending time. They can be stored in a database like SQLite. The OPC UA also allows to set up *AggregateConfiguration* objects and *AggregateFunctions*. These are used to aggregate data values and reduce the data load. It can be, for example, an average temperature aggregated from several sensors. Another functionality allows to set up data filtering, monitoring of changes, modification of history, etc. The HistoricalAccessConfiguration object contains general information about the historical data including the functionality that was used.

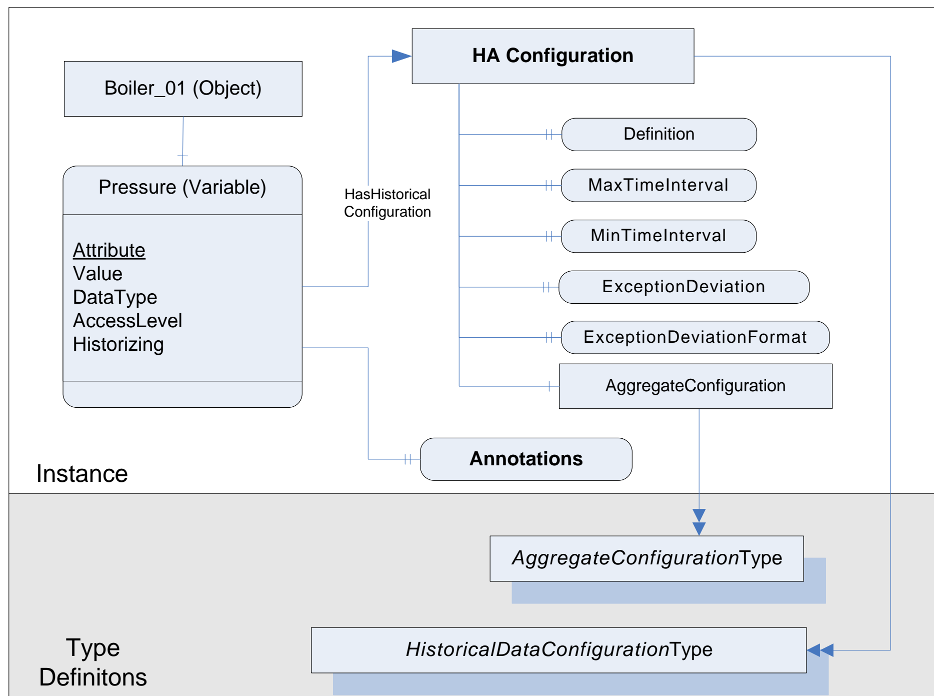


Figure 2.11: The data model of a DataVariable with history [8].

Services, Communication, and Security

OPC UA provides a *Service-Oriented Architecture* (SOA) for industrial applications [25]. It means that a complex application is composed of a set of independent components that offer functionalities as services. These services are accessible for other software components via a standard interface. Communication between particular applications in OPC UA is based on services accessible in a unified address space. This solution is advantageous because of its platform independence, universality and scalability.

The OPC UA specification defines a communication stack for either a client or a server implementation. Such implementation accesses the stack via the OPC UA API. It is not standardized, and therefore it may vary for different programming languages or operating systems. The OPC Foundation only standardized the way of communication. A client/server creates request messages based on the service definitions. A server then realizes a service and returns an appropriate response message. There is a technology mapping provided by OPC UA which says what mechanisms should be used on different communication layers. See the Figure 2.12 for the overview. The Part 6 of the specification [8] defines two technology mappings - *UA Native* and

UA Web Services. The first one supports only binary encoding, integrates TLS-like security mechanisms and it typically runs directly on TCP/IP. The second mapping allows both XML and binary encoding. When XML is used, the transmission speed is slower, but generic clients and various data interpretations are possible using SOAP[2] messages. The used network protocol is SOAP/HTTP(S). Other protocols can also be used but at the expense of interoperability [25].

The services in OPC UA are grouped into *Service Sets*. Table 2.5 shows an overview of these Service Sets. Brief descriptions are provided in the table no further elaboration is necessary for the purpose of this work. The important point is that services from these sets are used by client/server implementations.

Service Set	**Description**
Discovery	Services used to discover endpoints implemented by servers.
SecureChannel	Services that guarantee secure communication.
Session	Services for the connection and user access management.
NodeManagement	Services which manage to add and delete nodes.
View	Services that allow browsing the address space according to different views.
Query	Services for querying the address space.
Attribute	Services that enable reading and writing attributes.
Method	Services for calling methods.
MonitoredItem	Services that manage returning data for a subscription.
Subscription	Services used to manage subscriptions and data receptions.

Table 2.5: Overview of Service Sets.

Whenever a new software system is planned to be developed, a security assessment has to be performed and taken into account at the design stage. Part 2 of the OPC UA specification serves as a guideline for software designers and developers to ensure secure communication in developed systems. Since OPC UA interconnects the plant floor networks with higher level enterprise networks and standard HTTP ports are used in SOA implementations, the system can get vulnerable to some potential attacks from outside. An attacker can potentially get across several proxy servers. The following goals should be addressed.

[2]Simple Object Access Protocol is based on message exchanges in XML format mainly via HTTP. It is a base layer for the communication between web services.

- Authentication - Clients, servers, and users have to prove their identities (digital certificates or user credentials).
- Authorization - Access permissions can only be given to entities that truly need them. Actions they can perform should be limited.
- Confidentiality - Data encryption algorithms have to be used.
- Integrity - Receivers must get the same information that was sent.
- Auditability - All actions happening in a system must be recorded (tracking user activity).
- Availability - Systems have to be prevented from being overwhelmed.

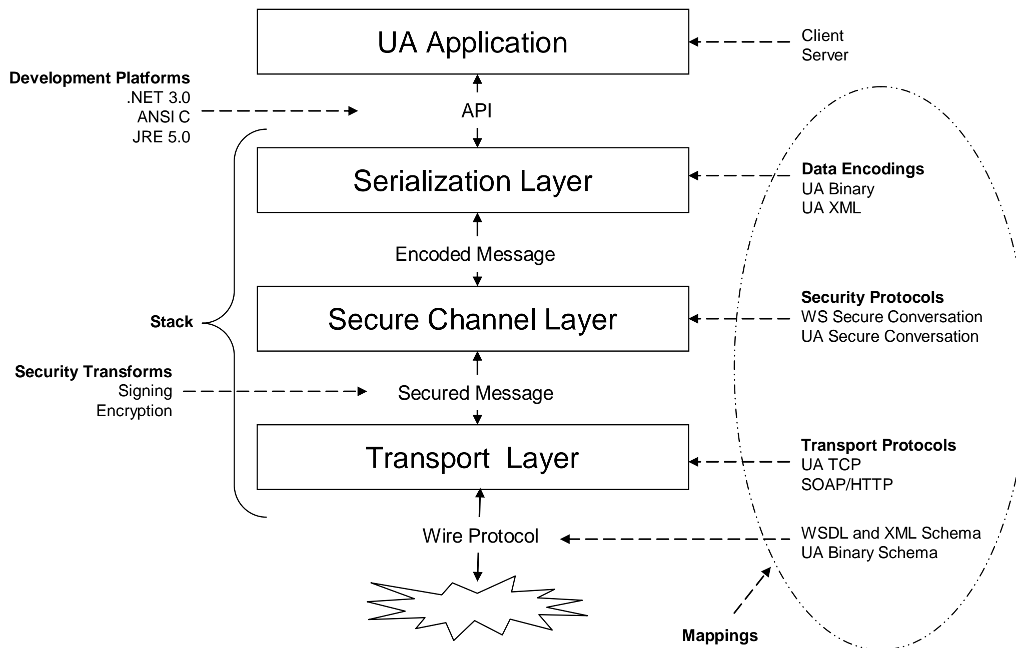


Figure 2.12: The OPC UA stack overview [8].

The security mechanism proposed by OPC UA is based on *Public Key Infrastructure* (PKI). A secure connection is established using *Asymmetric Cryptography* that uses a private and public key (standard X.509). Each application has a list of trusted public keys that represent trusted applications. Those keys are used to validate that the signature on a received message was generated by a corresponding private key. The certificates (keys) are issued by a Certificate Authority, or self-signed certificates can be used.

Another proposed option to secure the communication is to use WS Secure Conversation[3] which reduces the overhead of key establishment. This method uses *Symmetric Cryptography* and it is also recommended for OPC UA Web Services in [27] where authors elaborate the speed of encryption/decryption in a web browser.

[3]Web Services Secure Conversation enables the establishment of a secured session for long-running message exchanges. It uses a symmetric cryptographic algorithm which has better performance than the asymmetric one.[26]

Chapter 3

Existing OPC UA Tools and Implementations

This chapter reviews some of the existing tools and implementations that were taken into account for the development of the OWA. They were tried out, and some of them were also used during the development. Since OPC UA is nothing new, a lot of effort has already been given to the development of various applications.

3.1 OPC UA Testing Servers

Prosys OPC Ltd. company offers several OPC UA products [28]. There are applications like *Modbus Server*, *Historian*, *Gateway*, or developer tools like *SDK for Java*, *SDK for Delphi*, *C/C++ SDKs*, *.NET SDK*, and *Modeler*. Apart from these commercial products, this company also offers free test tools. More specifically, the tools are *Prosys OPC UA Simulation Server*, *Prosys OPC UA Client*, *Prosys OPC UA Client for Android*, and *Prosys OPC UA Client Pro Beta*.

The Prosys OPC UA Simulation Server tool was tried out. The tool is written in Java and has an intuitive user interface (Figure 3.1). When the tool is started on a computer, it automatically starts an OPC UA server which can be accessed via UA TCP or HTTPS. In the address space, there is a folder called Simulation in which a user can create new variables. Data values for

these variables are simulated by the server according to a user configuration. Unfortunately, that is all a user can do. It is not possible to create a custom structure of nodes with user-defined properties and relationships [29].

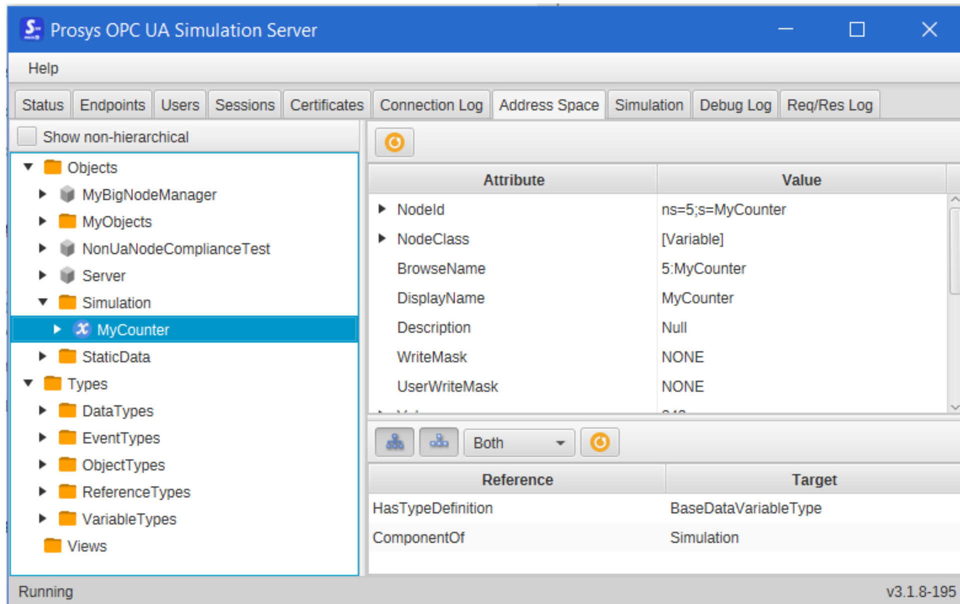


Figure 3.1: Graphical user interface of Prosys OPC UA Simulation Server.

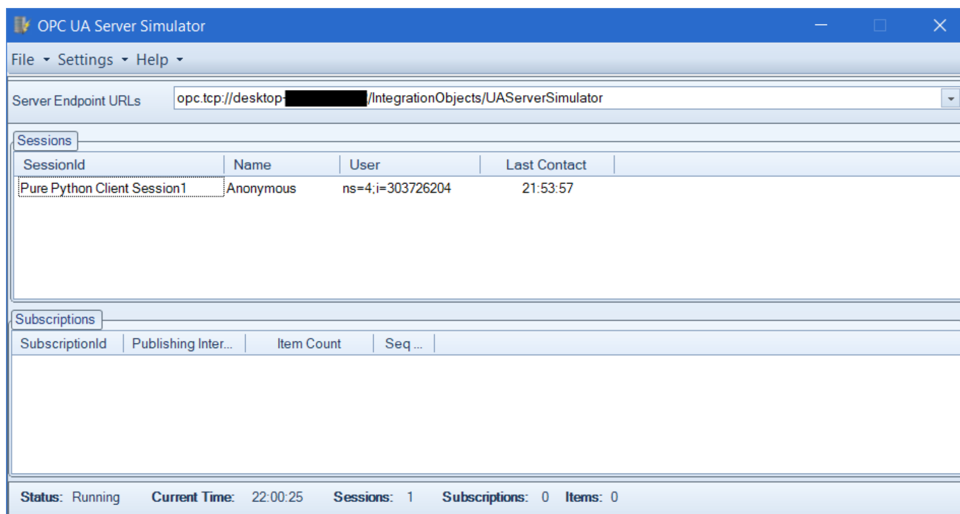


Figure 3.2: Graphical user interface of OPC UA Server Simulator from Integration Objects.

Another promising tool which was found is from Integration Objects companyv[30]. This company focuses on various OPC products like servers, clients, toolkits, and free tools. They provide *OPC UA Server Simulator* for free. It is a lightweight application that starts an OPC UA server and builds its address space from simple CSV files. These files define variables, their

types, values, and historical values. Access to the server can be observed from a graphical user interface (Figure 3.2). Unfortunately, like in the previous example, only variables can be defined and no other nodes in the address space.

Other companies like Unified Automation GmbH, Microsoft, or MathWorks also provide some OPC UA tools. Nevertheless, they are mostly not free, or their functionality is minimal.

3.2 OPC UA Web-based Clients

Besides the simulation server, Prosys OPC Ltd. company also offers *OPC UA Web Client* for accessing an OPC UA server and its data. The architecture of this tool can be divided into three layers as shown in Figure 3.3. The user interface in a web browser is a single-page application that uses AngularJS Javascript framework. It is designed in the form of a general browser of the tree structure on an OPC UA server. There is a service layer based on Node.js API and OPC UA Java SDK. The user interface communicates with the service layer via a *REST* API. REST stands for Representational State Transfer and it is an interface architecture style which is data-oriented and allows to create, read, edit, and delete data on a server using simple HTTP requests (GET, PUT, POST, DELETE). The service layer implements OPC UA client functionality which manages connections with OPC UA servers. [9].

One-Way Automation company has developed *OPC UA Web-Based Client* (Figure 3.4). It is a cross-platform web application that allows discovering OPC UA servers using Local Discovery Server (LDS), connect to them, browse their address space, monitor data values, and subscribe for data changes. This tool is developed in C++ using OPC UA C++ SDK from One-Way Automation and some third-party C++ libraries. The demo version can be tried out online [31], but otherwise, this product has to be purchased. The architecture model is similar to the previous example. Again, there is a web server that ensures the interconnection between the website and OPC UA servers. The company also mentions on its website that other future products are under consideration. One of them is called *REST adapter for OPC UA Servers*. It would allow having a specific user interface connected via a REST API to a middleware that would be responsible for further connection to OPC UA servers.

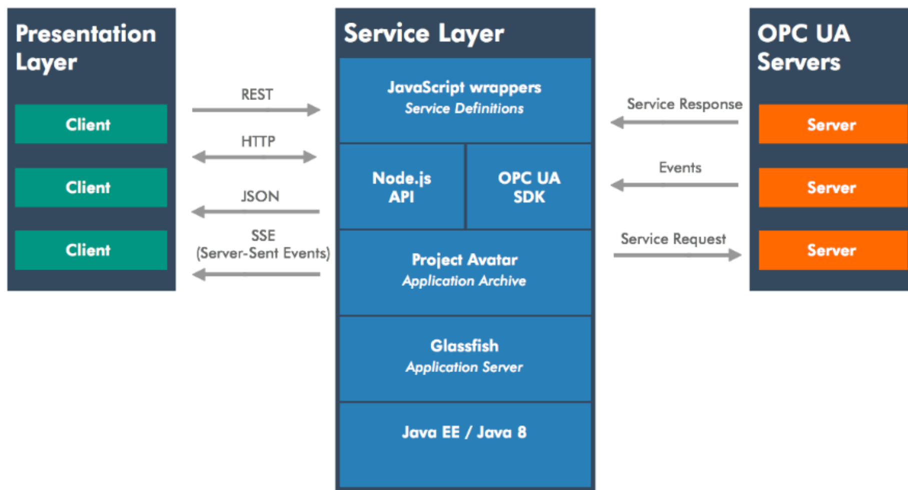


Figure 3.3: Architecture of Prosys OPC UA Web Client [9].

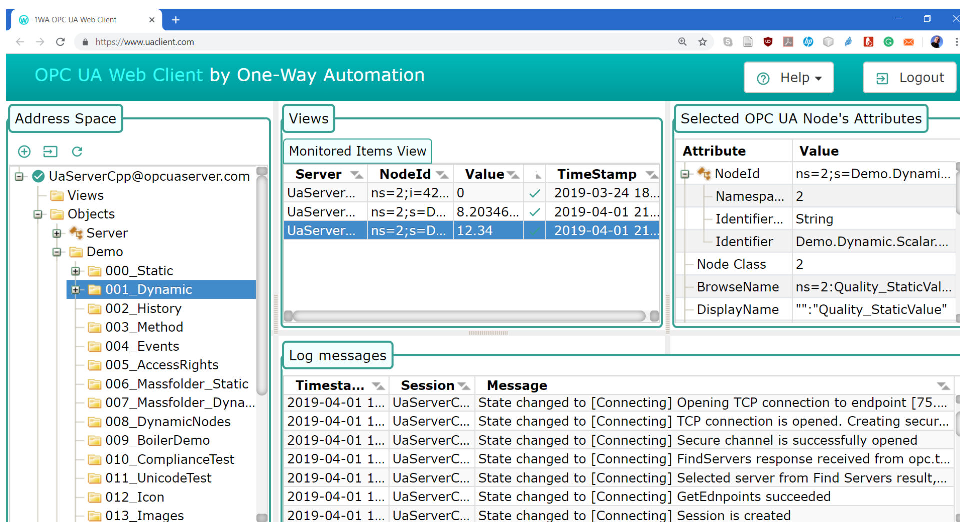


Figure 3.4: OPC UA Web Based Client developed by One-Way Automation.

3.3 OPC UA Software Libraries and Tools

There exist several software development kits (SDKs) for rapid development of OPC UA client and server applications. They are provided as software components to be reused in component-based software engineering (CBSE) process. Some of them are commercial, and some are open source. The commercial ones, which are called commercial off-the-shelf (COTS) components, are offered by companies like Integration Objects, Unified Automation,

National Instruments, or MathWorks. The commercial software components usually come with additional documentation and professional support.

Open source OPC UA implementations for both client and server in various programming languages are listed in Table 3.1. They usually differ in the implemented functionality, ease of use, and quality of documentation. Unfortunately, based on the research that was carried out, the majority of these software components are weakly documented, and they do not provide all features as proposed in the OPC UA specification. For example, most of them do not support XML protocol in terms of HTTP/SOAP.

Name	**Language**	**License**
open62541	C	MPL-2.0
UA.NET Standard	C#	GPL
node-opcua	JavaScript	MIT
FreeOpcUa	C++, Python	LGPL
ASNeG	C++	Apache
Eclipse Milo	Java	Eclipse Public License
S2OPC	C	Apache
opcua	Rust	MPL-2.0

Table 3.1: Open source OPC UA client and server implementations [13] available for commercial use.

Regarding the ease of use and implemented functionality, the *FreeOpcUa* project [10] seemed to be promising. This project consists of a few independent parts (libraries). The library *freeopcua* is written in C++ whereas *python-opcua* and *opcua-asyncio* are written in Python. The latter one is based on *asyncio* library and brings support for asynchronous behavior. Asynchronous programming makes code simpler, safer (lower chance for bugs), and has potentially better performance. Other parts within this project implement a client graphic user interface or a modeler tool.

It is worth mentioning that the *Free OPC UA Modeler* tool (Figure 3.5) of the FreeOpcUa project allows creating specific node structures that can be exported to an XML file. Such file can then be imported by a simple OPC UA server implementation that can run a customized OPC UA server. As an example, Listing 3.1 shows two exported nodes in the XML format. The first node is an object and the second node is a variable of the object.

```
<UAObject BrowseName="1:ledStates" NodeId="ns=1;i=14301"
    ↪ ParentNodeId="ns=1;i=14300">
    <DisplayName>ledStates</DisplayName>
    <Description>LED states of the controller</Description>
    <References>
      <Reference ReferenceType="HasTypeDefinition">ns=1;i=8009</
    ↪ Reference>
      <Reference ReferenceType="HasProperty">ns=1;i=8242</
    ↪ Reference>
      <Reference ReferenceType="HasComponent">ns=1;i=14302</
    ↪ Reference>
    </References>
  </UAObject>
  <UAVariable BrowseName="1:LED1" DataType="Byte" NodeId="ns=1;i
    ↪ =14302" ParentNodeId="ns=1;i=14301">
    <DisplayName>LED1</DisplayName>
    <Description>LED1</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">ns
    ↪ =1;i=14301</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference
    ↪ >
    </References>
    <Value>
      <uax:Byte>1</uax:Byte>
    </Value>
  </UAVariable>
```

Listing 3.1: Example of how the exported OPC UA server structure may look.

Further OPC UA related tools and implementations for various use cases can be found over the Internet. For example, the company Software Toolbox, Inc. offers many innovative products. This company published a real-world case study [32] which resulted in implementation and deployment of an OPC Server for consuming RESTful JSON web services.

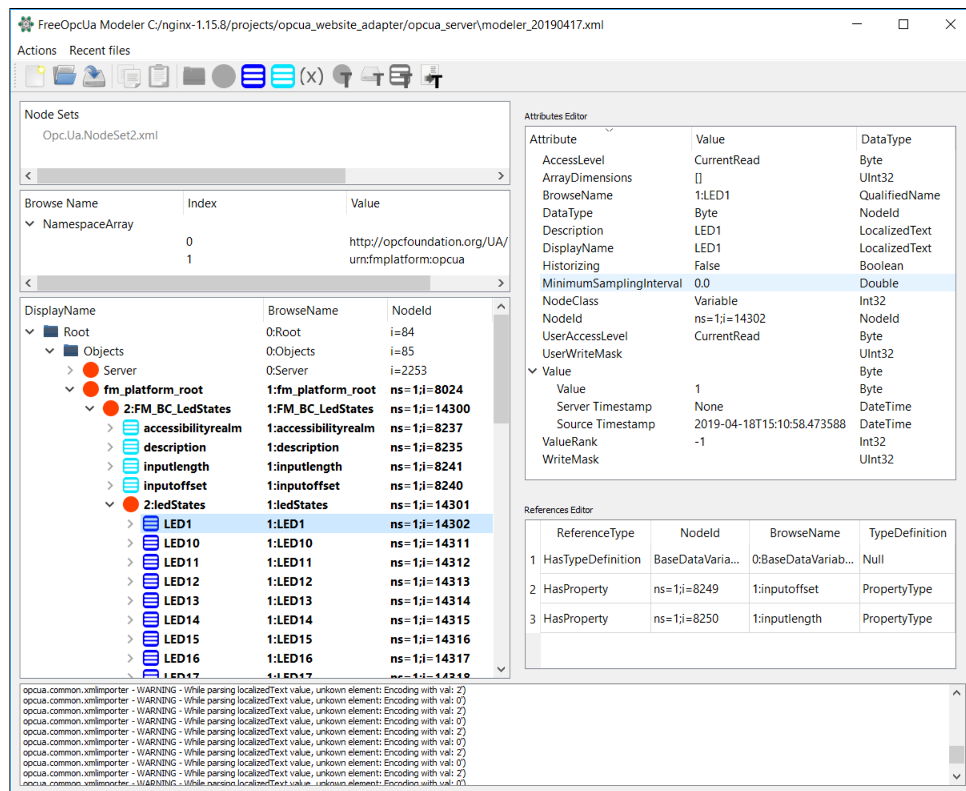


Figure 3.5: Illustration of the Free OPC UA Modeler tool [10].

The Legacy Software Architecture

This chapter is concerned about the architecture of the legacy software solution as well as its components and data structures that were supposed to be used together with the OWA application. It serves as an analysis of the existing solution, especially the user interface and metadata model.

The former architecture of the diagnostics system consists of the following main components: *Web Client Application*, *BaseWeb*, *Web Server Application*, *HistoryDB Service*, and *Diagnostic Service*. The Web Client Application is a user interface tailored for an operator to view the system diagnostics. The BaseWeb component is responsible for authentication, it provides static content like HTML, CSS, and Javascript files, it forwards requests to the Web Server Application, it provides an interface for HTTP/WebSocket communication, and it handles socket events. The Web Server Application handles specific diagnostic functions.

The Web Server Application provides metadata which describes the semantics of the diagnostic data that are being obtained from controller modules. Both data and metadata can be accessed via a REST API which also allows the Web Client Application to request a call of remote procedures. The diagnosed controller modules can be, for example, a computer that controls a level crossing, railroad switch, or signal light. The HistoryDB service allows to store and read historical diagnostic objects. The Diagnostic Service maintains the communication with *Diagnostic Router* which tracks any data changes.

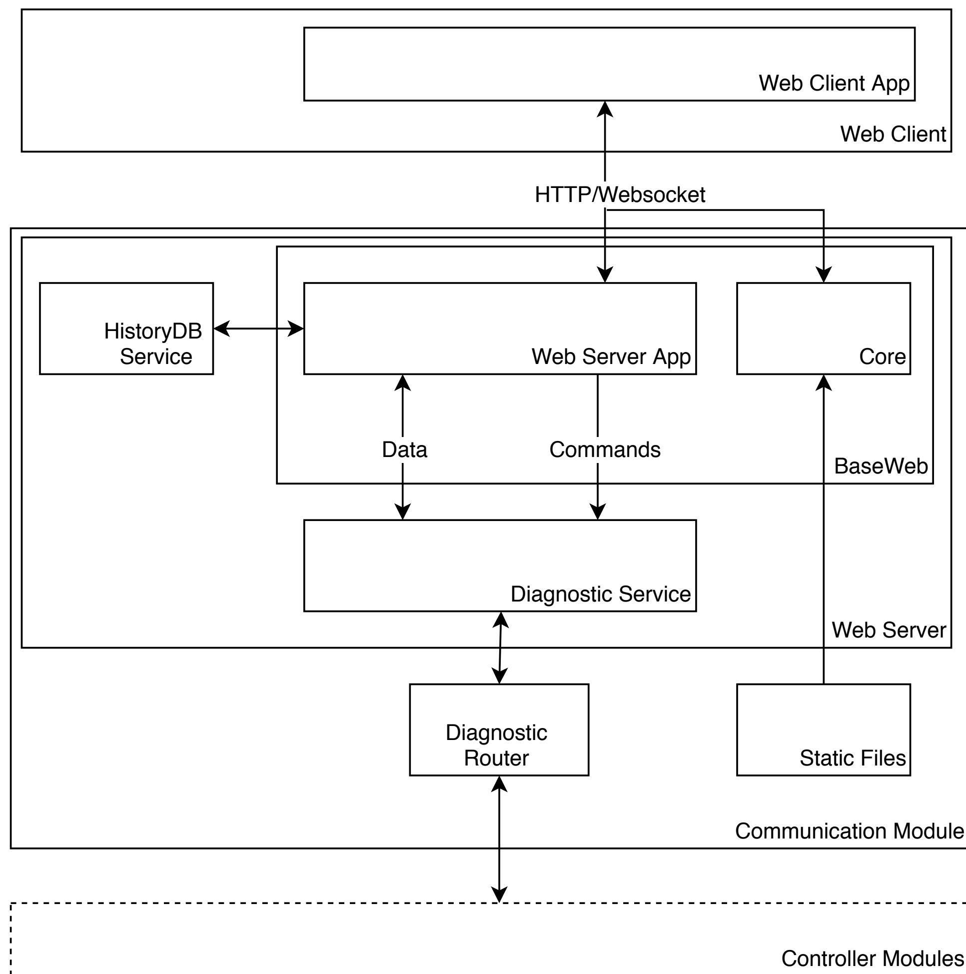


Figure 4.1: Block diagram of the legacy architecture.

4.1 User Interface

The existing user interface is a single-page [33] modular Javascript application which communicates with a corresponding web server via a REST API. The API documentation was reviewed and summarized to a well-arranged table of endpoints with expected HTTP requests and responses (see the example in Appendix B).

After the web page is loaded, it firstly asks for metadata in XML files. These files describe the data model of the diagnosed system. Then it attempts to retrieve several objects from the web server to initialize the user interface. If those objects are not provided, the web page does not finish loading. Details about those objects are listed in Table 4.1.

ID	Name	Description
1	AllDiagnosis	Root element of all diagnosis objects.
13012	LedStatus	LED states of the controller
14002	SwCode	Software version
10544	MAC0	MAC address of eth0
10545	MAC1	MAC address of eth1
21663	NICAddrIPv4	IPv4 address of the network interface
21664	NICAddrIPv6	IPv6 address of the network interface

Table 4.1: Initial objects requested by the web page.

Figure 4.2: The user interface of Signal Control Module (SCM) as an example of a Web Client Application that was supposed to use the OWA for interconnection with a target OPC UA server.

The application also tries to establish a WebSocket connection. Communication via the WebSocket connection is crucial for the majority of transferred data which is asynchronously received from the web server.

An example of how the user interface can look like is shown in Figure 4.2. The issue is a user interface of so-called Signal Control Module (SCM) which

controls light signals on a railroad. The Front Panel follows the physical arrangement of the target device.

4.2 Metadata

The data model used on the client side which is described by metadata called *Client Schema* had been designed in a previous product version and needed to be retained as far as possible. Any changes in the existing metamodel would result in changes in the Web Client Application. Real names of the Client Schema instances as XML elements are written in capital letters in this document. The metamodel is composed of OBJECTs with properties (attributes) and object components. In the Client Schema, the components are called PROPERTIES. Each OBJECT can have other OBJECT and PROPERTY elements nested in the PROPERTIES element. Both OBJECT and PROPERTY elements can have various properties. In the XML file, they can be encoded either as XML attributes of XML elements or as other XML elements nested in their parent XML elements. Those properties which are encoded as XML elements can have their properties as XML attributes. An example is a property DESCRIPTION which has an attribute LANGUAGE. The metadata model is better described in Figure 4.3.

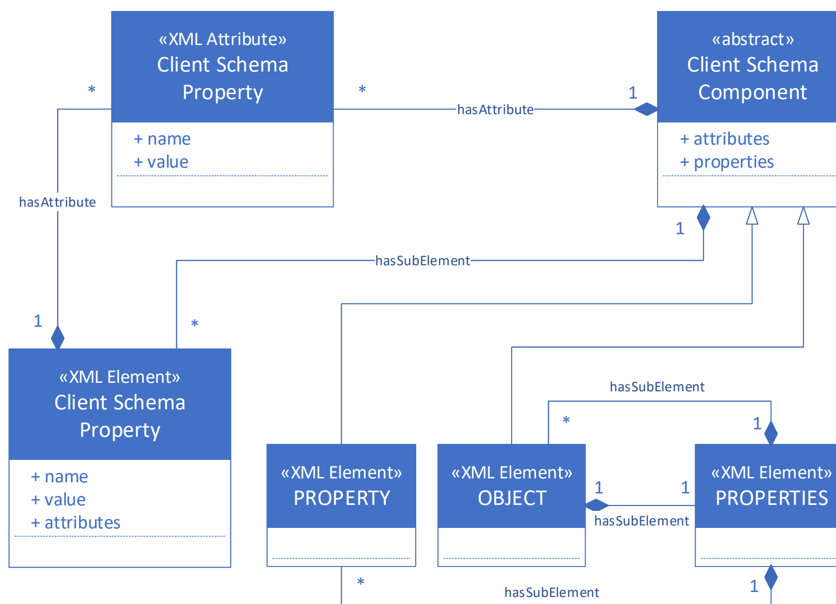


Figure 4.3: Meta model of the Client Schema (structure of XML content). Capital letters are used in literal names of XML elements.

Chapter 5

Design and Implementation

The main goal for this thesis was to design and develop a software solution, called the OPC UA Website Adapter (OWA), that would interconnect the existing web-based user interface with the OPC UA server introduced into a new version of a safety-critical railway controller. Before the development was started, the review of the literature and various resources had been performed. Previous chapters were concerned about it. This chapter is focused on how the assigned problem was solved with the aid of the literature review findings.

5.1 Motivations and Design Decisions

Chapter 2 reviewed generally the communication architectures in industry and more specifically in the railway industry. It was shown that the state-of-the-art reference architecture for railway signaling systems employs the OPC UA as a protocol for the diagnosis network (see Figure 2.8 for more details). It is one of the reasons why the existing software solution regarding the diagnostics needed to be upgraded. The new version of railway controllers that was being developed was supposed to follow this reference architecture.

Although there exist many different OPC UA tools, none of them provided such flexibility, that it could be adjusted to work with the existing user interface. The most promising project that was discovered was the intended REST adapter for OPC UA Servers from One-Way Automation. Unfortunately, it had not been developed yet by the time of working on this

thesis. It follows that a special tool needed to be developed to solve the problem - *to interconnect the existing web-based user interface with the OPC UA server introduced in the new safety-critical railway controller.* Chapter 3 introduced some existing OPC UA tools that could be utilized during software development. The Free OPC UA Modeler was found most useful since it could be used for modeling of a proofing OPC UA server. Also, the FreeOpcUa library in Python was chosen as the most appropriate software development kit because of the required application portability and simplicity.

Chapter 4 introduced how the diagnostic software architecture looked like in the obsolete version. The software components used on the backend side of the old system were found useless because of its inner complexity and estimated difficulty of adjusting to the new requirements. Also, the architecture needed to be simplified and adjusted to modern standards. These facts spoke for replacement of the existing backend solution which meant to design and implement a new one.

No general OPC UA client could replace the existing web-based user interface. The user interface needed to be customized to ensure a user-friendly interface for an employee as a system operator. The existing web-based user interface could be altered so that the connection to the OPC UA server could be established directly via HTTP/SOAP. However, this option would mean a significant data load since all data is sent in XML format when SOAP is used. Also, securing the connection is more challenging as discussed in [27].

The authors in [34] proposed an extension of the OPC UA with RESTful services. The REST architectural style has many advantages and is appropriate for decentralized applications. It proposes the use of a fixed service interface to transfer diverse resource representations. This approach improves universality, independence, and reusability. Thanks to the use of the REST API in the old diagnostics system, the user interface could be preserved. The OWA application had to follow this architectural style. As a result, it could be envisioned as a RESTful gateway to diagnostics of the subsequent system. One instance of the OWA application could potentially be connected to more OPC UA servers to collect more complex diagnostics data. Also, another decentralized OPC UA server could store a load of historical values as was recommended in the OPC UA specification [8].

Figure 5.1 shows a potential deployment of the OWA. Each Element Controller contains an OPC UA server, and the OPC UA adapter incorporates an OPC UA client. The Human-Machine Interface is an application that calls particular endpoints of the REST API of the web server within the OWA.

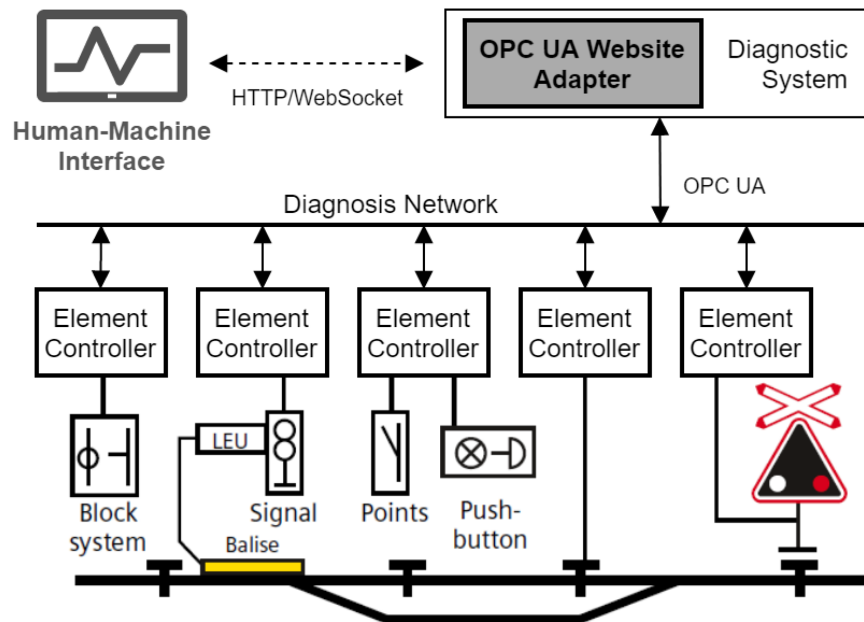


Figure 5.1: Prospective deployment of the OPC UA Website Adapter.

5.2 Requirements Analysis

This section summarizes the basic requirements for software development. The requirements were developed according to the Master's Thesis Assignment, discussions with involved experts from Siemens Mobility, s.r.o, and personal expert judgment. The requirements in Table 5.1 and Table 5.2 already expect that the OWA application must consist of an OPC UA client and a web server. The key roles of the OWA application are:

1. connection with the target OPC UA server,
2. semantic translation (different data model of the OPC UA server and the Web Client Application),
3. data acquisition from the OPC UA server, and
4. API provision to the Web Client Application including provision of the data obtained from the OPC UA server.

Code	Description
FR1	The OWA shall include a web server component that shall provide the following API endpoints for HTTP requests as the old diagnostic system: ▪ GET */metadata* (description of the metamodel) ▪ GET */objects* (historical values of variables) ▪ GET */objects/properties* (actual values of variables) ▪ GET */files* (engineering files) ▪ GET,POST */commands* (remote procedures)
FR2	The web server shall enable the data to be sent in the following formats: binary, XML, and JSON. The desired format shall be decoded from each HTTP request (from the extension of a required resource).
FR3	The web server shall provide an endpoint (GET */objects/*) for negotiation of a WebSocket connection.
FR4	The WebSocket connection shall serve a web client with actual data values that were changed on the target OPC UA server. Only the binary format shall be used for this purpose.
FR5	The OWA shall include a component of OPC UA client that shall be responsible for connection with an OPC UA server according to a given configuration loaded from a file.
FR6	The OPC UA client shall use a certificate and a private key for authentication and data encryption when communicating with the OPC UA server.
FR7	After a successful connection, the OPC UA client shall browse the target server and export its metamodel to an XML file.
FR8	After a successful connection, the OPC UA client shall subscribe for any data changes on the target OPC UA server.
FR9	After a successful connection, the OPC UA client shall perform a metadata mapping to convert the metamodel of the OPC UA server to the metamodel understandable for a connected web client.
FR10	The metadata mapping process shall follow a set of rules loaded from an external file. It shall facilitate any later changes and extensions.
FR11	The OWA shall store the history of all data received from the OPC UA server within a current application run and up to a limit defined in an external configuration file.
FR12	The OPC UA client shall be capable of calling remote procedures on the OPC UA server when requested from a web client.

Table 5.1: Functional requirements

Code	Description
NFR1	The existing user interface from the old diagnostic system shall be reused with no or minimal changes.
NFR2	The REST API defined in FR1 which specifies the communication between a web client and the OWA shall be clearly described within this project.
NFR3	The web server shall be responsive, and requests should be handled asynchronously.
NFR4	The design of the OWA shall retain a component-based approach with event-driven architecture.
NFR5	To ensure fast historical data access, data shall be held in hashed collections.
NFR6	The metadata mapping shall be designed in a way so that it can be easily changed or extended.
NFR7	The OWA shall be well documented by UML diagrams.
NFR8	The developed source code shall be well commented.
NFR9	A sample OPC UA server data model shall be designed, and a simple test OPC UA server shall be implemented for testing purposes.
NFR10	Git version-control system shall be used for tracking of changes in the source code.
NFR11	The OWA as a standalone application shall be portable for both Linux and Windows operating systems.

Table 5.2: Non-functional requirements

5.3 OPC UA Test Server

Since no OPC UA server had been designed by the time of working on this thesis, a test server had to be created for testing purposes. It was designed according to the provided metadata from the legacy software. The test server data model was created in the Free OPC UA Modeler which was reviewed in Chapter 3. An example of the structure is demonstrated in Figure 5.2. Then a simple OPC UA server implementation was created in Python using the FreeOpcUa library.

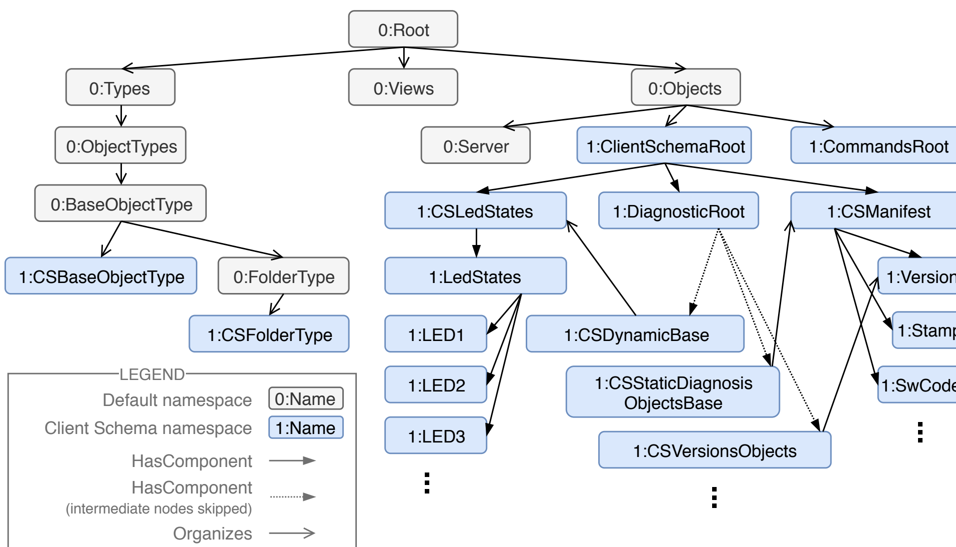


Figure 5.2: Example of how the test server is modeled according to the original metadata structure. Note that only *HasComponent* and *Organizes* relationships are displayed. In reality, there are many other relationships among the nodes. For example, the node *DiagnosticRoot* has a relationship *HasTypeDefinition* with the node *CSFolderType*.

5.4 Metadata Mapping

There exist two different data models. The first one is unified and describes the data structure on an OPC UA server. The second one is proprietary and describes the data that can be sent to the Web Client Application. It was discussed in Section 4.2. The problem was that to interconnect the OPC UA server with the Web Client Application, these two data models needed to be mapped so that the data representations would be compatible. The advantage was that both data models are object-based. The problem of data models mapping can be separated into data mapping and metadata mapping.

The data mapping is a process of creating links between data elements that may need to be aggregated, separated, or converted to different formats. In this case, the actual data elements did not need to be aggregated nor separated. The data formats in both data models are identical except for *DateTime* and *ByteString* OPC UA data types. Data values of these data types have to be converted in the OWA application. Table 5.3 shows all data type mappings.

OPC UA Data Type	Size [B]	Client Schema Data Type	Size [B]
Boolean	1	BOOL	1
SByte	1	INT8	1
Byte	1	UINT8	1
Int16	2	INT16	2
UInt16	2	UINT16	2
Int32	4	INT32	4
UInt32	4	UINT32	4
Int64	8	INT64	8
UInt64	8	UINT64	8
Float (IEEE 754)	4	-	
Double (IEEE 754)	8	-	
String (UTF-8)	1/char	STRING (UTF-8)	1/char
DateTime (LDAP/FILETIME)	8	UINT32	4
ByteString (binary)	1/byte	ARRAY (Base64)	1/byte

Table 5.3: Mapping of data types between OPC UA and the proprietary Client Schema format.

The two data models have different metadata and a mapping needed to be developed. It started with an analysis of the relationships between nodes in the OPC UA tree structure and between XML elements in the web client metadata. A systematic approach was applied to link OPC UA nodes to XML elements of the web client metadata. Each OPC UA node has attributes that can be directly converted to attributes of an XML element. Then it has a set of references to other nodes. These references have to be sorted by their names, directions, and information about the referenced nodes. The problem of this mapping can be seen as a linking of tuples *(SourceNodeClass, ReferenceName, ReferenceDirection, TargetNodeClass, TargetNodeDisplayName)* to XML elements or XML attributes of these elements. A lookup table had to be created to define the links. It was realized by a JSON object that is self-describing, easy to understand, and can be loaded from an external file. An example of the developed JSON object structure is shown in Appendix C.

To visually demonstrate the metadata mapping in diagrams, the approach from [35] (an extension of UML language) was loosely adapted. The picture in Figure 5.3 shows the mapping on the table level. A more detailed description of the mapping algorithm is provided in Listing 5.1.

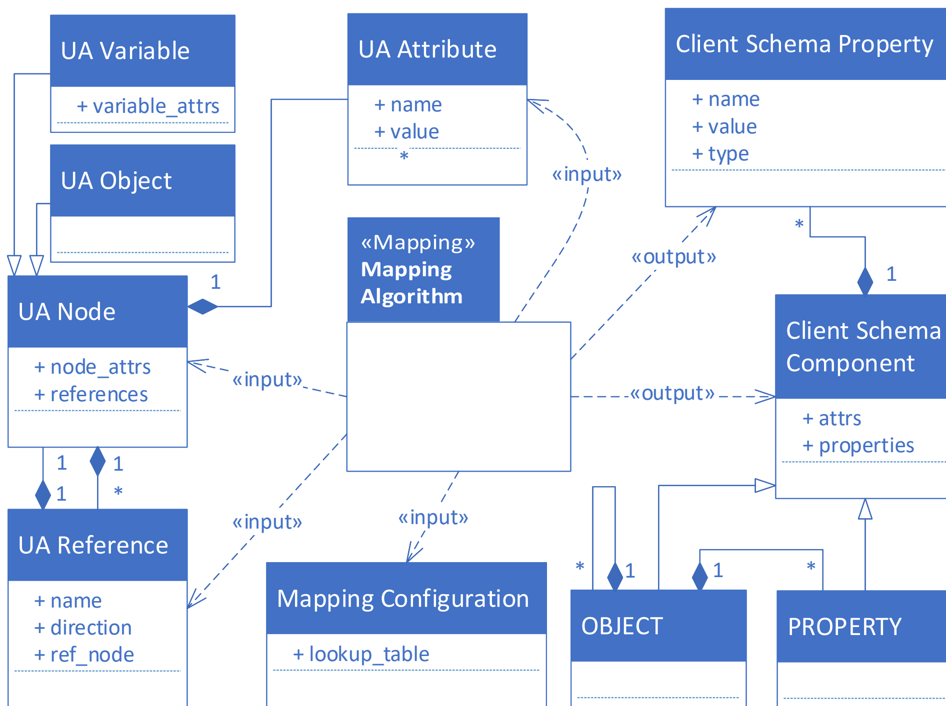


Figure 5.3: Mapping diagram showing the metadata mapping problem.

```
config := load_config()
opcua_tree := load_opcua_tree()
xml_tree.root = map_node(opcua_tree.root)

FUNCTION map_node(node)
    // mapping of attributes
    xml_elem := new()
    map_attrs(node, xml_elem)

    // mapping of references
    FOR ref_type, ref_node IN node.get_references()
        mapping = config.get_mapping(node,ref_type,ref_node)
        IF mapping != null THEN
            IF mapping.is_expandable() THEN
                xml_elem.add_sub_elem(map_node(ref_node))
            ELSE
                map_attrs(ref_node,xml_elem)
            END
        END
    END
    return xml_elem
END
```

```
FUNCTION map_attrs(node, xml_elem)
    FOR attr IN node.get_attributes()
        mapping = config.get_mapping(node,attr)
        IF mapping != null THEN
            xml_elem.add_attribute(mapping.name, attr.value)
        END
    END
END
```

Listing 5.1: Pseudocode of the mapping algorithm.

5.5 Software Design

To summarize the critical functions of the OWA application, Figure 5.4 presents the use case diagram. The main actors are the Web Client and the OPC UA Server. The OWA application provides API for the Web Client and ensures metadata mapping, secure communication, data collection, and handling of remote procedure calls.

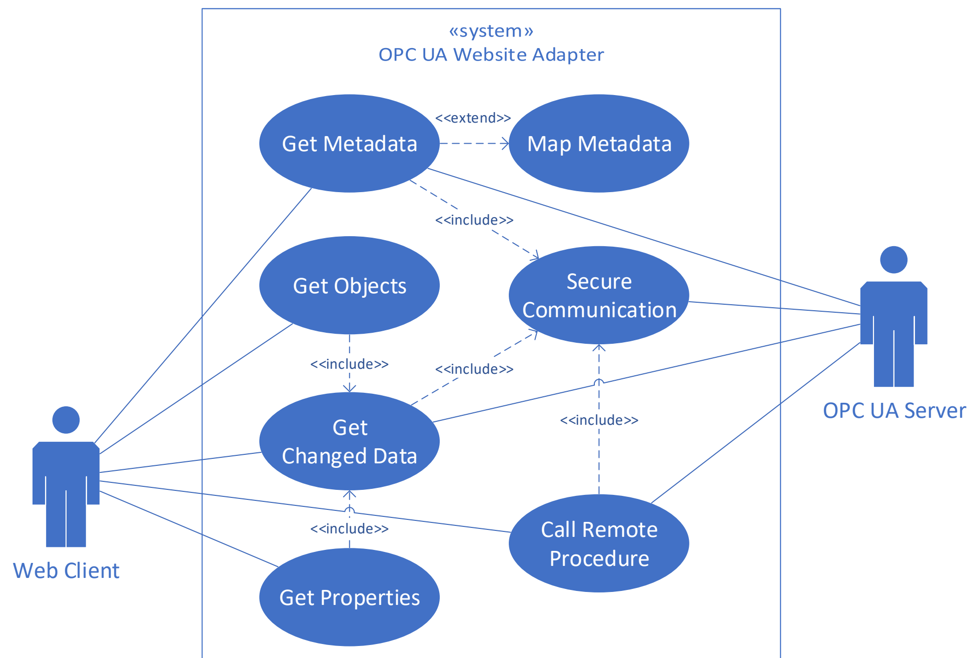


Figure 5.4: UML Use Case diagram for the OWA application.

The OWA application was designed to consist of two main components - *OPC UA Client* and *Web Server*. The Web Server component was supposed to provide the same REST API as the web server in the legacy software described in Chapter 4. The OPC UA Client component, on the other hand, uses OPC UA Services provided by an OPC UA Server to get diagnostic data. Finally, the components are interconnected via the bidirectional Data Service API. Within this interface, the Web Server can ask the OPC UA Client for data or metadata. It can also require a remote procedure call. The OPC UA Client notifies the Web Server about new data values whenever they change on the OPC UA Server to which the client is subscribed. The relationships are shown in Figure 5.5.

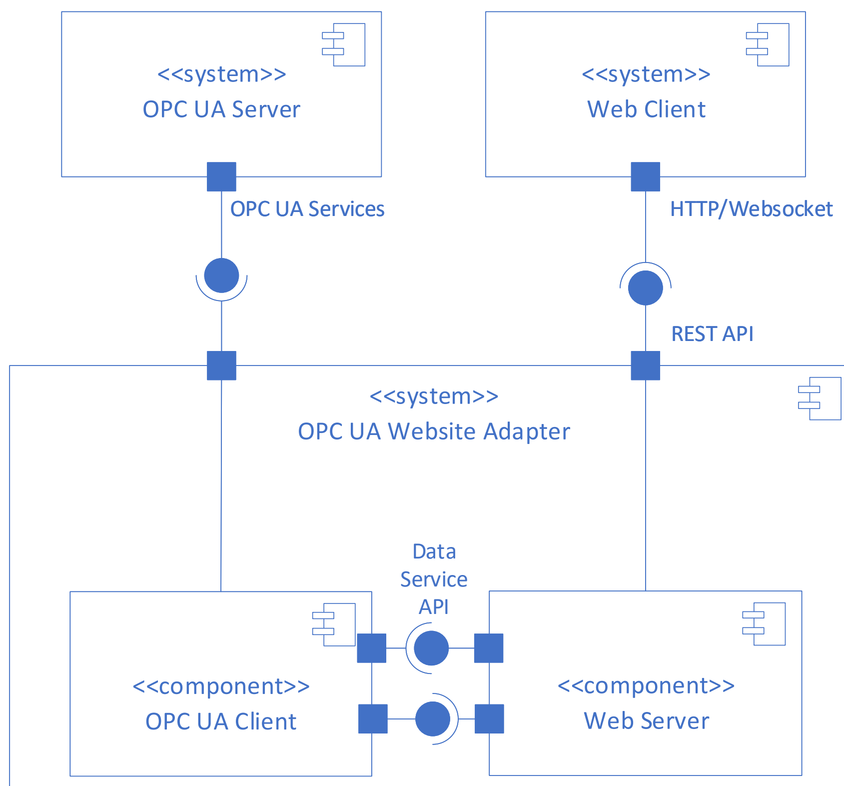


Figure 5.5: UML Component Diagram for the OWA application.

The Web Client and the OWA were supposed to be deployed on the same computer as shown in the UML Deployment Diagram in Appendix D. The design presented in Figure 5.5 allows the system to be easily deployed differently. However, since no other deployment model was required, the

channel between the Web Client and the Web Server did not have to be secured. However, the communication between the OPC UA Client and the OPC UA Server was required to be secured. The OPC UA protocol has its services for Secure Channel establishment which exploit *Asymmetric Encryption*. Each computer with the OWA and the Web Client has to have a certificate which implies subsequent user access rights.

Afterward, the behavior between particular components and systems was designed. The OPC UA Client component starts as first, loads its configuration from a file, establishes a secured connection with the OPC UA Server, and performs metadata mapping. Finally, it subscribes for any changes in all data variables on the server. Then it initializes the Web Server Component which also loads its configuration and starts listening for incoming requests. When the Web Client is connected, it may ask the Web Server for metadata, objects, or properties. It can also request a WebSocket connection or ask for calling a remote procedure. To have actual data to be sent via the WebSocket, the Web Server must observe the OPC UA Client component. Therefore, the Web Server with the OPC UA Client was supposed to implement the observer pattern. Figure 5.6 shows the UML Sequence Diagram.

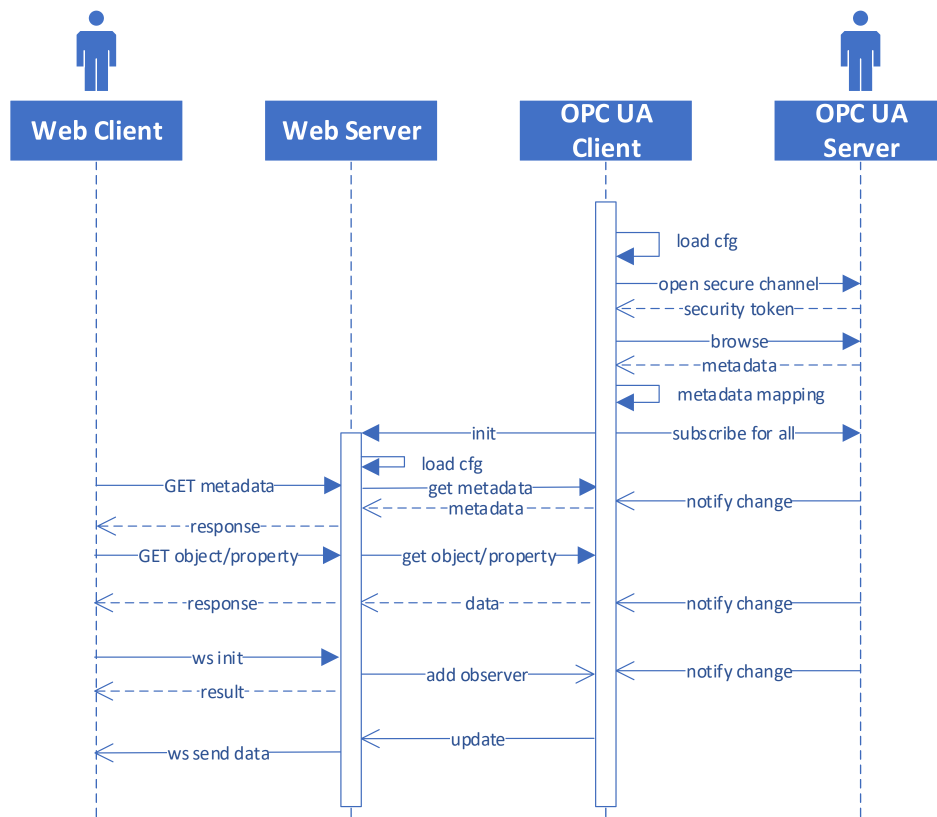


Figure 5.6: UML Sequence Diagram for the OWA application.

5.6 Implementation

The OWA application was implemented in Python as a console application. The software architecture follows the design which was described in the previous section. The OPC UA Client component is based on the FreeOpcUa library (python-opcua version), and the *AIOHTTP* framework was used to develop the Web Server. The UML Class Diagram in Appendix E gives an overview of the details of the implementation.

The class *OWA_OPCUAClient* wraps the general OPC UA client implementation from the library. This class initializes the Client object and calls its methods for connection, target server browsing, and data acquisition.

Regarding the server browsing, the OPC UA client firstly analyzes the target server structure. All node identifiers are grouped in a hashmap collection which tracks the *parent-child* relationships. Each key represents a parent node ID, and a corresponding list of values represents children of the node. This servers as a fast look-up table, when all data values belonging to a certain node and its children in the tree structure are required.

Secondly, the OPC UA client instigates the process of building XML metadata from the target server structure using the *ClientSchemaBuilder* class. This class performs metadata mapping. Each OPC UA node has attributes and references which are evaluated according to the mapping configuration file. The file records say which attributes (or references with attributes of corresponding referenced nodes) should be translated to which XML elements. For details, see Appendix C. Each record contains a template that is accompanied by attribute values from a respective OPC UA node. The resulting string is parsed to one or more XML elements which then compose the final XML tree that is serialized to a file.

Another important part of the OWA_OPCUAClient class is *DataHandler*. There are functions like *datachange_notification()* or *event_notification()* that are configured as callback functions for the FreeOpcUa library. They are called as a result of data changes on the server. The DataHandler takes care of aggregating the changed data, inserting them into a history queue (Figure 5.7) and notifying all registered observers. The queue aggregates data values of properties (referenced by identifiers) to points in time (referenced by indexes). The properties in history can be accessed via indexes. When the queue size reaches a configured limit, the least recent values are popped and discarded. The most recent properties can be accessed in a hashmap which keeps the latest values for all available properties.

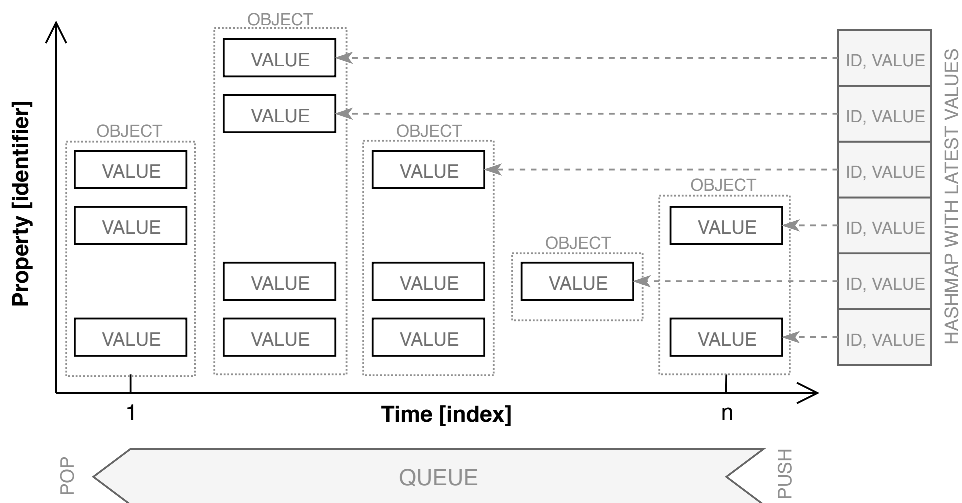


Figure 5.7: Visualization of the queue that stores changed property values in time. Each set of property values at a given point in time is referred to as objects.

The Web Server is formed by the *Application* class of the AIOHTTP library and its *UrlDispatcher*. The *EndpointsHandler* consists of functions that are linked with the UrlDispatcher, and they handle incoming HTTP requests. The EndpointsHandler is responsible for preparation of a corresponding HTTP response which carries the data obtained from the DataHandler. It includes transformation of data to a desired format - binary, XML, or JSON. When a WebSocket connection is required from a client, the data values are forwarded after the DataHandler notifies the EndpointsHandler as its observer. The connected client can also require properties from history via the WebSocket. This kind of live connection demands a well-synchronized access to the constantly shifting history queue. Binary semaphores are used in this case.

5.7 Library fix

The FreeOpcUa library [10] did not support *LocalizedText* which was crucial for correct working of multilingual names and descriptions in the metadata. Therefore, the functionality was added to the library. Changes were made in the files listed below.

- /opcua/ua/uatypes.py
- /opcua/common/manage_nodes.py
- /opcua/common/xmlimporter.py
- /opcua/common/xmlparser.py

The paths are relative from the installation folder of Python packages. The corrected files are part of the thesis attachments. The original files in the *opcua* Python package need to be substituted with the corrected ones for the OWA to work properly.

5.8 Deployment

The OWA application was implemented in Python (version 3.7.2) which implies that it can be deployed on any computer with Python environment. The programming constructs used in the implementation are supported in Python 3.6 and higher. As was mentioned in the previous sections, the OWA application is based on several Python libraries which need to be installed before running the application. It can be achieved by running `pip install -r requirements.txt`. All required dependencies are listed in *requirements.txt* file which is distributed with the application.

Note that `pip3` has to be used on Linux and likewise the `python3` or `python3.6` command. After installing the dependencies, the corrected files of the FreeOpcUa library have to be copied into a subsequent installation folder as mentioned in Section 5.7.

The developed source code files were packed into a zip archive which is executable in Python. It can be run as shown in Listing 5.2. The app needs a configuration file in .json format. This file with default configurations is distributed together with the executable archive. Other necessary folders and files are created when the application is started. If a secured connection is required, subsequent certificate and private key have to be placed in */files/security* subfolder.

```
C:\projects>python opcua-website-adapter-0.1.zip
INFO - __main__ - OPC UA Website Adapter started.
INFO - owa_opcuaclient - Analyzing target OPC UA server...
INFO - owa_schema_builder - Mapping metadata...
INFO - owa_web - Running web server on http://localhost:8081 ...
INFO - owa_web - Keyboard interrupt.
INFO - owa_web - Web server closed.
INFO - owa_opcuaclient - OPC UA client terminated.
INFO - __main__ - OPC UA Website Adapter terminated.

C:\projects>
```

Listing 5.2: Example of running the OWA application

5.9 Validation

The software was tested as a whole system, and it was done manually. Creation of automated tests was not in the scope of this thesis. The Table 5.4 briefly describes how the particular functional requirements were validated.

Finally, the original user interface was tested with the OWA application and the connected OPC UA test server with modeled data. The deployment was tested on Windows 10 with Python 3.7 and Ubuntu 18.04 with Python 3.6.

Code	Description of the test approach
FR1, FR2	The API endpoints were tested using Postman software for API development. The software supports all formats used in the OWA application (.json, .xml, .bin) for sending/receiving body of requests/responses.
FR3, FR4	The WebSocket connection was tested from the console of Google Chrome web browser using the Javascript web application from the old diagnostic system.
FR5	The OPC UA client firstly loads a configuration file and then initiates the connection to a server with addresses given in the file. If the configuration file cannot be loaded or the server is not found, a corresponding error message is logged. This behavior was tested, and the log file was inspected.
FR6	The connection was tested with encryption that used an example certificate and private key provided by the FreeOpcUa library. The communication failed when the used certificate or key was not valid.
FR7	The XML file with exported nodes by the OPC UA client was compared with the XML file that is loaded by the test OPC UA server to build its structure.
FR8	The test OPC UA server contained a routine that was continuously changing the value of one data variable in the predefined value range and period. A WebSocket connection with the OWA was established, and incoming messages were observed to validate a correct behavior.
FR9, FR10	The test OPC UA server was modeled according to the original metadata used by the Web Client Application. The algorithm that performs the metadata mapping takes nodes and references as its input and produces an XML file with Client Schema metadata. The output file was compared with the original metadata.
FR11	The component of the OPC UA client contains a queue as described in Section 5.6. The queue size is configured from the configuration file. Using the Postman software, requests for objects were generated for indexes in the range from 0 to the configured value.
FR12	The test OPC UA server contained a test method which was run as a result of a request from the Postman software.

Table 5.4: Validation of functional requirements

Conclusion

Within this work, communication in distributed industrial systems was studied with a focus on diagnostics. The evolution of Fieldbus protocols prefaced the OPC UA standard that was theoretically examined in more detail. It introduced how diagnostic data is organized and accessed uniformly. Also, the present state-of-the-art reference architecture for distributed control systems was reviewed. It was found out that railway signaling systems follow the same direction as industry regarding the decentralization of controllers along railroads. The latest reference architecture designed by the Eulynx initiative showed that the OPC UA protocol is an inherent part of modern railway signaling systems.

The original solution of system diagnostics was analyzed and documented as part of the preparation for a new solution. The original web server serving data for the client application was found incapable of adaptation to new standards and requirements. However, the initial Web Client Application was retained in its original form.

The main contribution of this work is the design and implementation of the OPC UA Website Adapter. It was proposed as a portable software tool which can be envisioned as a RESTful gateway to diagnostics of underlying systems like controllers of semaphores, level crossings, etc. The developed solution provides the same API as the old diagnostic system allowing the former user interface to be used. In the new diagnostic system which utilizes the OPC UA standard, the diagnostic data is stored in a different data model. The OWA application analyzes the data structure on a target OPC UA server and performs metadata mapping. The server structure is mapped onto the

data model used by the original Web Client Application. The application interacting with a system operator can then ask for diagnostic data or send commands to respective control systems.

For testing and demonstration purposes, Human-Machine Interface of the *Point Control Module* (PCM) was chosen because it required the least amount of test data to be modeled on the test server. Nevertheless, more than 300 nodes in the OPC UA address space had to be created for the Web Client Application to load and work in limited mode.

The implementation exploits the FreeOpcUa library in Python programming language. Unfortunately, the library does not fully support some functionality defined by the OPC UA specification. The problem was especially with the *LocalizedText* data type and the *ListOfLocalizedText* that is crucial for labels in multiple languages. An incidental contribution of this thesis is a proposal on how this functionality could be easily added.

The concept of implementing a middle (service) layer between a web client and OPC UA server is not novel as showed the review of existing solutions. However, all other projects were too general and were found incapable of integration with the current user interface of this project.

By the time of submitting this thesis, the OWA application had been already employed in development of other diagnostic components in Siemens Mobility in Prague. As the future work, improvements of the FreeOpcUa library should be proposed to the library maintainers. Also, automated tests should be developed to ensure the correct working of the OPC UA Website Adapter. The application will most likely have to be extended with a mechanism of retrieving historical diagnostic data from another OPC UA server dedicated for this purpose. Alternatively, the queue that keeps diagnostic data up to its size limit could be extended with a database solution.

Appendix A

Bibliography

[1] R. Zurawski, Ed., *Industrial communication technology handbook*, second edition ed., ser. Industrial information technology series. Boca Raton: CRC Press, Taylor & Francis Group, 2015.

[2] P. Drahos, E. Kucera, O. Haffner, and I. Klimo, “Trends in industrial communication and OPC UA,” in *2018 Cybernetics & Informatics (K&I)*. Lazy pod Makytou: IEEE, Jan. 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8337560/

[3] T. Bangemann, S. Karnouskos, R. Camp, O. Carlsson, M. Riedl, S. McLeod, R. Harrison, A. W. Colombo, and P. Stluka, “State of the Art in Industrial Automation,” in *Industrial Cloud-Based Cyber-Physical Systems*, A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J. L. Lastra, Eds. Cham: Springer International Publishing, 2014, pp. 23–47. [Online]. Available: http://link.springer.com/10.1007/978-3-319-05624-1_2

[4] *IoT 2020: smart and secure IoT platform: white paper.* Geneva, Switzerland: International Electrotechnical Commission, 2016, oCLC: 973571117.

[5] Mobility Division, “Innovations Trackguard Sinet and Sigrid,” Siemens Switzerland Ltd., Wallisellen, Switzerland, Tech. Rep., Aug. 2014.

[6] ——, “Trackguard Sinet Annaberg-Buchholz Süd Pilot Project,” Siemens AG, Berlin, Germany, Tech. Rep., 2014.

[7] Maarten van der Werff, Frans Heijnen, and Mirko Blazic, “EULYNX: The Next Generation Signalling Strategy,” in *Aspect 2015*. London: Institution of Railway Signal Engineers, Sep. 2015, pp. 1–10.

[8] OPC Foundation, Inc., "OPC Unified Architecture Specification," Tech. Rep., Jul. 2012. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture

[9] Tatu Paronen, "Prosys OPC UA Web Client released," Aug. 2014. [Online]. Available: https://www.prosysopc.com/blog/prosys-opc-ua-web-client-released/

[10] Alexander Rykovanov and Oroulet, "Free OPC-UA Library." [Online]. Available: https://github.com/FreeOpcUa

[11] Fieldbus Inc., "IEC61158 Technology Comparison," Austin, Texas, USA, Oct. 2011. [Online]. Available: http://www.fieldbusinc.com/downloads/fieldbus_comparison.pdf

[12] "SmartRail 4.0 Subconcept Transfer System," SmartRail 4.0 collaboration, Bern, Switzerland, Tech. Rep., Jul. 2018. [Online]. Available: https://smartrail40.ch/

[13] "Open source implementation of OPC UA (OPC Unified Architecture) aka IEC 62541 licensed under Mozilla Public License v2.0: open62541/open62541," Apr. 2019, original-date: 2013-12-20T08:45:05Z. [Online]. Available: https://github.com/open62541/open62541

[14] K. Schwab, *The fourth industrial revolution*, first u.s. edition ed. New York: Crown Business, 2016.

[15] Dave Evans, "The Internet of Everything: How More Relevant and Valuable Connections Will Change the World," Cisco Systems, Inc., San Jose, CA, USA, Tech. Rep., 2012. [Online]. Available: https://www.cisco.com/c/dam/global/en_my/assets/ciscoinnovate/pdfs/IoE.pdf

[16] P. Ryznar, "The Future Of Factories," *Forbes*, 2018. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/06/19/the-future-of-factories/

[17] OPC Foundation, "OPC Foundation - The Industrial Interoperability Standard." [Online]. Available: https://opcfoundation.org

[18] Mike Wasson, "ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET," Nov. 2019. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[19] Evan You, "Vue.js." [Online]. Available: https://vuejs.org/

[20] Max Felser, "The fieldbus standards: History and structures." University of Applied Science Berne, 2002.

[21] J. R. Moyne and D. M. Tilbury, "The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 29–47, Jan. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4118467/

[22] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8401919/

[23] J. Smith, "ERTMS - European Rail Traffic Management System," Sep. 2016. [Online]. Available: https://ec.europa.eu/transport/modes/rail/ertms_en

[24] M. Heinrich, J. Vieten, T. Arul, and S. Katzenbeisser, "Security Analysis of the RaSTA Safety Protocol," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. Miami, FL: IEEE, Nov. 2018, pp. 199–204. [Online]. Available: https://ieeexplore.ieee.org/document/8587371/

[25] S.-H. Leitner and W. Mahnke, "OPC UA - Service-oriented Architecture for Industrial Applications," *Softwaretechnik-Trends*, vol. 26, no. 4, 2006. [Online]. Available: http://pi.informatik.uni-siegen.de/stt/26_4/01_Fachgruppenberichte/ORA2006/07_leitner-final.pdf

[26] IBM Knowledge Center, "Web Services Secure Conversation," Oct. 2014. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cwbs_wssecureconv.html

[27] A. Braune, S. Hennig, and S. Hegler, "Evaluation of OPC UA secure communication in web browser applications," in *2008 6th IEEE International Conference on Industrial Informatics*. Daejeon, South Korea: IEEE, Jul. 2008, pp. 1660–1665. [Online]. Available: http://ieeexplore.ieee.org/document/4618370/

[28] Prosys OPC, "Prosys OPC - Software Products." [Online]. Available: https://www.prosysopc.com/

[29] ——, "OPC UA Simulation Server User Manual," Prosys OPC Ltd., Espoo, Finland, Tech. Rep. Version 3.1.6, Sep. 2018. [Online]. Available: https://downloads.prosysopc.com/opcua/apps/JavaServer/dist/2.2.2-109/Prosys_OPC_UA_Simulation_Server_UserManual.pdf

[30] I. Objects, "OPC UA Server Simulator Quick User Guide," Integration Objects, Houston, Texas, Tech. Rep. Version 1.0 Rev.0, May 2018.

[31] One-Way Automation, "1wa OPC UA Web Client," Jun. 2018. [Online]. Available: https://www.uaclient.com/

[32] Software Toolbox, Inc., "Innovative OPC Solution for Web Services Minimizes Customer Infrastructure Costs while Maximizing Data Access." [Online]. Available: https://support.softwaretoolbox.com/ci/fattach/get/87525/1441896808/redirect/1/filename/SWTB_ConneXSoft_JSON_Avtech.pdf

[33] "Single-page application," Feb. 2019, page Version ID: 884239800. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=884239800

[34] S. Gruner, J. Pfrommer, and F. Palm, "RESTful Industrial Communication With OPC UA," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1832–1841, Oct. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7407396/

[35] S. Luján-Mora, P. Vassiliadis, and J. Trujillo, "Data Mapping Diagrams for Data Warehouse Design with UML," in *Conceptual Modeling – ER 2004*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.-W. Ling, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3288, pp. 191–204. [Online]. Available: http://link.springer.com/10.1007/978-3-540-30464-7_16

Appendix B

Example of the Web Server REST API

```
GET      /metadata.extension
------------------------------------------------------------------
    REQUEST:     HTTP/1.1
                 Content-Length: 0

    RESPONSE:    HTTP/1.1 200 OK
                 Content-type: application/extension
                 Content-length: number

                 (list of files in XML or JSON)

GET      /metadata
------------------------------------------------------------------
    REQUEST:     HTTP/1.1
                 Content-Length: 0

    RESPONSE:    HTTP/1.1 400 Bad Request
                 Content-Length: 0

GET      /metadata/filename.extension
------------------------------------------------------------------
    REQUEST:     HTTP/1.1
                 Content-Length: 0

    RESPONSE:    HTTP/1.1 200 OK
                 Content-type: application/extension
                 Content-Length: number

                 (file content)

GET      /objects/count
------------------------------------------------------------------
    REQUEST:     HTTP/1.1
```

```
                Content-Length: 0
    RESPONSE:   HTTP/1.1 200 OK
                Content-Type: application/octet-stream
                Content-Length: 8

                uint64

GET     /objects/count.extension
------------------------------------------------------------------
    REQUEST:    HTTP/1.1
                Content-Length: 0
    RESPONSE:   HTTP/1.1 200 OK
                Content-Type: application/extension
                Content-Length: number

                <count value="count_value"/>
                or
                {"count": count_value}

GET     /objects/index
------------------------------------------------------------------
    REQUEST:    HTTP/1.1
                Content-Length: 0
    RESPONSE1:  HTTP/1.1 200 OK
                Content-Type: application/octet-stream
                Content-Length: number

                raw_object

    RESPONSE2:  HTTP/1.1 410 Gone
                Cache-Control: no-cache
                Content-Length: 0
    (The object is not available and the index is less than or
    ↪ equal to the objects count.)

    RESPONSE3:  HTTP/1.1 404 Not Found
                Content-Length: 0
    (The object is not available and the index is greater than the
    ↪  objects count.)

    RESPONSE4:  HTTP/1.1 400 Bad Request
                Content-Length: 0
    (The index is equal to 0.)

GET     /objects/
------------------------------------------------------------------
    REQUEST:    HTTP/1.1
                Content-Length: 0
                Connection: Upgrade
                Upgrade: websocket
    RESPONSE:   HTTP/1.1 WebSocket Protocol Handshake
                Connection: Upgrade
                Upgrade: WebSocket

                index raw_object raw_object index ...
```

Appendix C

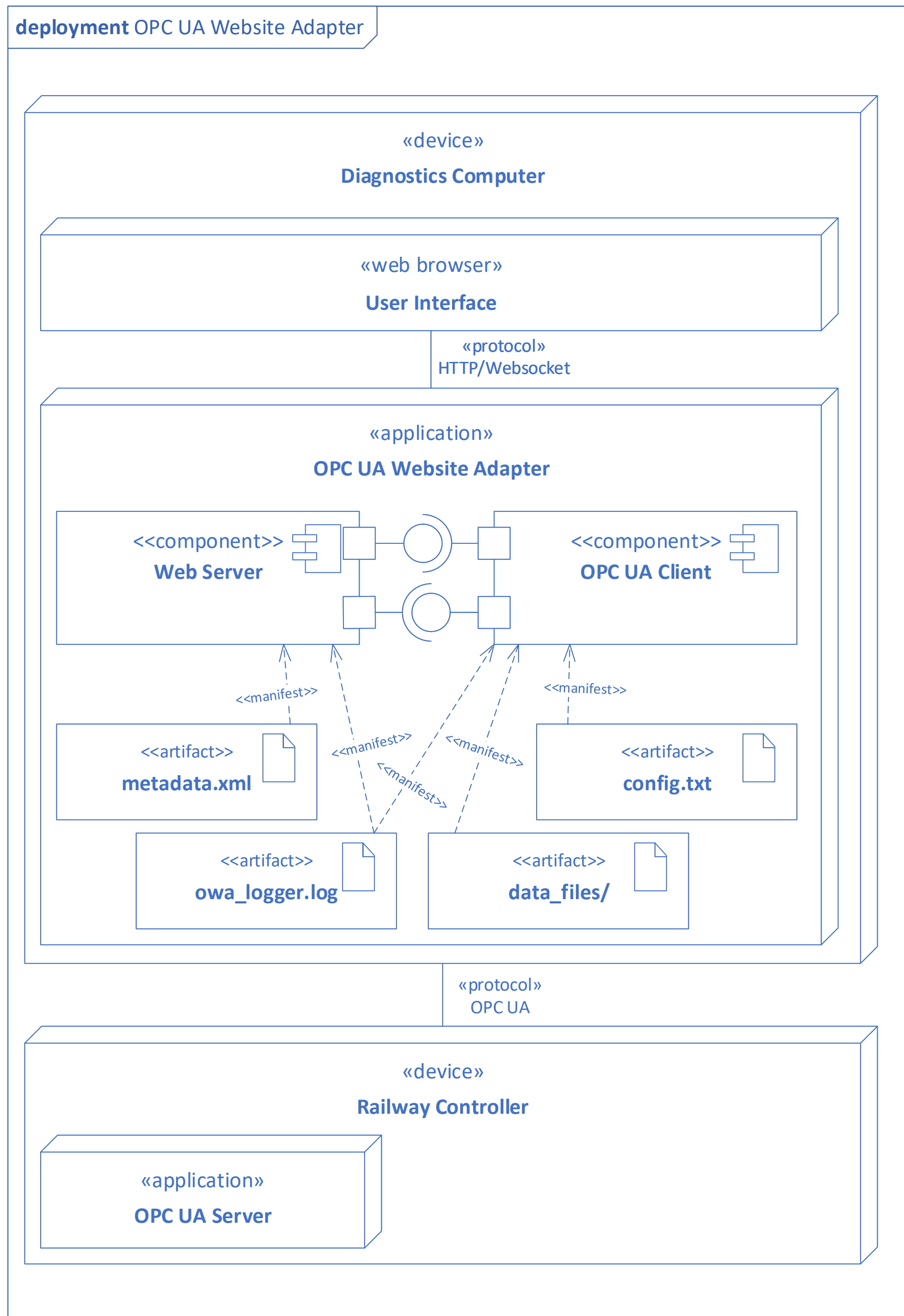
Example JSON Object for Metadata Mapping (Lookup Table)


```
{
 "SrcNodeClass": {
  "Object": {
   "AttributeMappings": [
    {
     "AttributeId": 1,
     "ValueCount": 1,
     "SrcRegEx": ";i=(.*)\\)$",
     "DstType": "attr",
     "DstTemplate": "id=\"{}\""
    },
    {
     "AttributeId": 4,
     "ValueCount": 1,
     "SrcRegEx": ", Text:(.*)\\)$",
     "DstType": "elem",
     "DstTemplate": "<techname>{}</techname>"
    }
   ],
   "References": [
    {
     "Direction": "Forward",
     "ReferenceName": "HasProperty",
     "ReferenceId": 46,
     "DstNodeDisplayName": {
      "constant": {
       AttributeMappings": [
        {
         "AttributeId": 13,
         "ValueCount": 1,
         "SrcRegEx": "(.*)",
```

```
          "DstType": "attr",
          "DstTemplate": "constant=\"{}\""
         }
        ]
       },
       "description": {
        "AttributeMappings": [
         {
          "AttributeId": 13,
          "ValueCount": 2,
          "SrcRegEx": "^LocalizedText\\(Encoding:3, Locale:(.*),
    ↪ Text:(.*)\\)$",
          "DstType": "elem",
          "DstTemplate": "<description language=\"{}\">{{}}</
    ↪ description>"
         }
        ]
       }
      }
     },
     {
      "Direction": "Inverse",
      "ReferenceName": "Organizes",
      "ReferenceId": 35,
      "DstNodeDisplayName": {
       "*": {
        "AttributeMappings": [
         {
          "AttributeId": 1,
          "ValueCount": 0,
          "SrcRegEx": ";i=(.*)\\)$",
          "DstType": "elem",
          "DstTemplate": "<parents></parents>",
          "HasChildElem": "reference"
         },
         {
          "AttributeId": 1,
          "ValueCount": 1,
          "SrcRegEx": ";i=(.*)\\)$",
          "DstType": "elem",
          "DstTemplate": "<reference parentid=\"{}\"/>",
          "HasParentElem": "parents"
         }
        ]
       }
      }
     }
    ]
   },
   "Variable": {
    ...
   }
  }
}
```

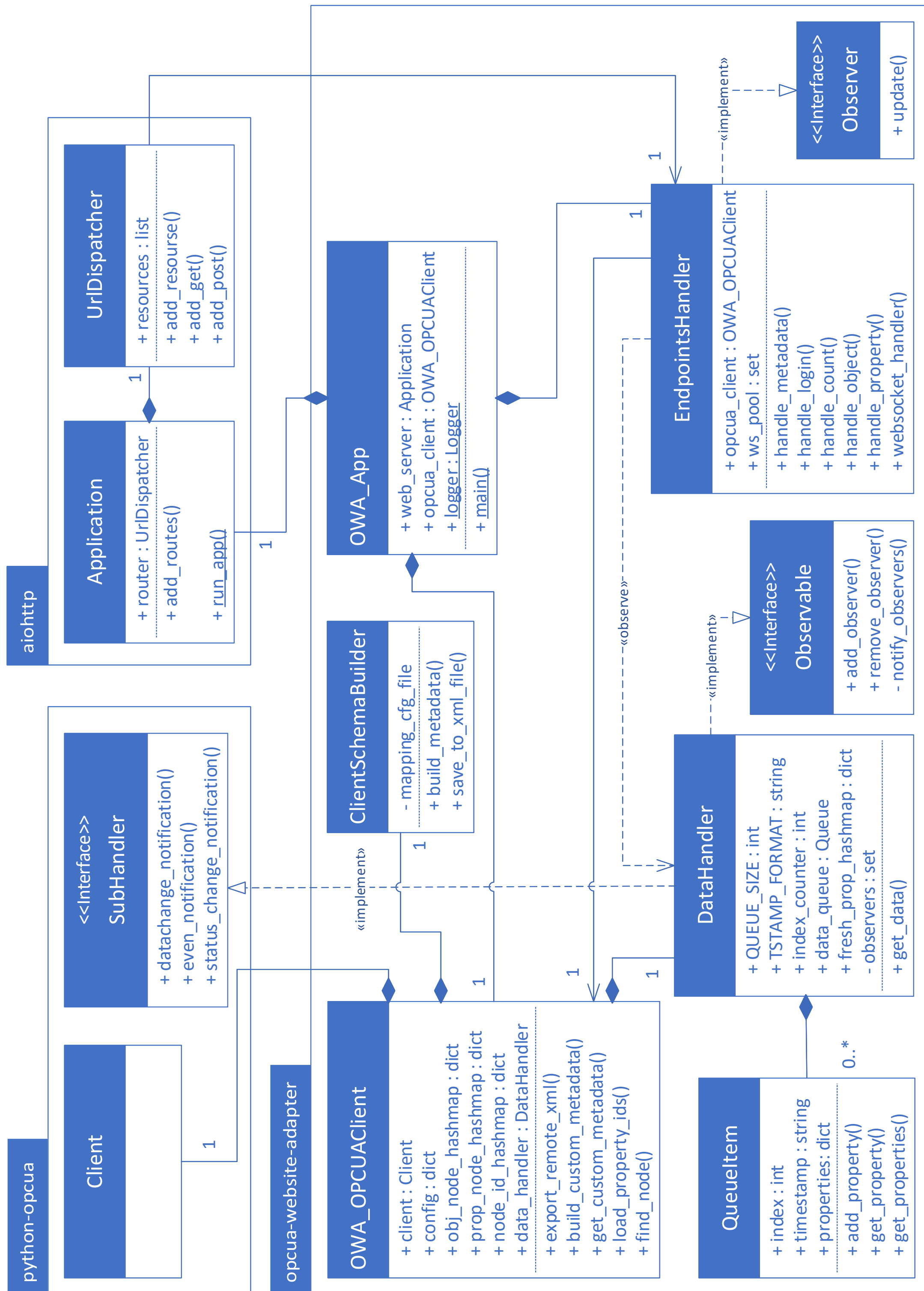
Appendix D

UML Deployment Diagram


deployment OPC UA Website Adapter
«device»
Diagnostics Computer
«web browser»
User Interface
«protocol»
HTTP/Websocket
«application»
OPC UA Website Adapter
<<component>>
Web Server
<<component>>
OPC UA Client
<<manifest>>
<<manifest>>
<<manifest>>
<<manifest>>
<<manifest>>
<<artifact>>
metadata.xml
<<artifact>>
config.txt
<<artifact>>
owa_logger.log
<<artifact>>
data_files/
«protocol»
OPC UA
«device»
Railway Controller
«application»
OPC UA Server

Appendix E

UML Class Diagram



aiohttp
Application
+ router : UrlDispatcher
+ add_routes()
+ run_app()
UrlDispatcher
+ resources : list
+ add_resourse()
+ add_get()
+ add_post()
python-opcua
Client
<<Interface>>
SubHandler
+ datachange_notification()
+ even_notification()
+ status_change_notification()
opcua-website-adapter
OWA_App
+ web_server : Application
+ opcua_client : OWA_OPCUAClient
+ logger : Logger
+ main()
ClientSchemaBuilder
- mapping_cfg_file
+ build_metadata()
+ save_to_xml_file()
OWA_OPCUAClient
+ client : Client
+ config : dict
+ obj_node_hashmap : dict
+ prop_node_hashmap : dict
+ node_id_hashmap : dict
+ data_handler : DataHandler
+ export_remote_xml()
+ build_custom_metadata()
+ get_custom_metadata()
+ load_property_ids()
+ find_node()
EndpointsHandler
+ opcua_client : OWA_OPCUAClient
+ ws_pool : set
+ handle_metadata()
+ handle_login()
+ handle_count()
+ handle_object()
+ handle_property()
+ websocket_handler()
<<Interface>>
Observer
+ update()
<<Interface>>
Observable
+ add_observer()
+ remove_observer()
- notify_observers()
DataHandler
+ QUEUE_SIZE : int
+ TSTAMP_FORMAT : string
+ index_counter : int
+ data_queue : Queue
+ fresh_prop_hashmap : dict
- observers : set
+ get_data()
QueueItem
+ index : int
+ timestamp : string
+ properties: dict
+ add_property()
+ get_property()
+ get_properties()
«implement»
«observe»
1
0..*