Kasperi Kiviluoma

Implementing OPC UA server into embedded device

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

1 December 2023

Abstract

Author:	Kasperi Kiviluoma
Title:	Implementing OPC UA server into embedded device
Number of Pages:	35 pages
Date:	1 December 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Smart IoT Systems
Supervisors:	Keijo Länsikunnas, Senior Lecturer Henri Auer, Software Team Leader

The purpose of this final year project was to explore the possibilities of the Open Platform Communications Unified Architecture standard and the best practices on how to integrate it into the case company's X-MET analyzer device. This was achieved by providing a prototype server application that integrates into the Linux environment of the analyzer.

The server application is based on the open62541 C library and the open62541pp C++ wrapper library. The libraries allowed to rapidly develop the server with a focus on demonstrating the features of the standard. The server application runs on the analyzer device and provides an access point for client software to connect to.

The required features for the prototype server application were achieved. As a result, the server application models the analyzer device in a format defined by the standard. It provides access to the data produced by the device with any client software that is compliant with the standard. The prototype server is used to demonstrate how the company's analyzer devices can be integrated into the Open Platform Communications Unified Architecture standard.

The findings of this project help the case company evaluate the possibilities of the Open Platform Communications Unified Architecture standard for their products. The next steps for the development of the server application would be to integrate the server more tightly into the hardware of the device and provide more of the features of the analyzer available for client applications.

Keywords: OPC UA, Embedded systems, XRF

Tiivistelmä

Tekijä:	Kasperi Kiviluoma
Otsikko:	OPC UA palvelimen sovittaminen sulautettuun laitteeseen
Sivumäärä:	35 sivua
Aika:	1.12.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Smart IoT Systems
Ohjaajat:	Lehtori Keijo Länsikunnas Ohjelmistotiimin johtaja Henri Auer

Opinnäytetyön tarkoituksena oli selvittää Open Platform Communications Unified Architecture -standardin mahdollisuuksia ja parhaita käytäntöjä sen integroimiseksi case-yrityksen X-MET-analysaattorilaitteisiin. Tämä saavutettiin kehittämällä prototyyppipalvelinohjelmisto, joka integroituu analysaattorilaitteen Linux-ympäristöön.

Palvelin perustuu open62541 C -kirjastoon ja open62541pp C++ -käärekirjastoon. Kirjastot mahdollistivat palvelimen nopean kehityksen keskittyen standardin ominaisuuksien esittelyyn. Palvelinohjelmisto toimii analysaattorilaitteessa ja tarjoaa yhteyspisteen yhteydenpitoon asiakasohjelmiston kanssa.

Prototyyppipalvelinohjelmisto täyttää sille asetetut toiminnallisuuteen liittyvät vaatimukset. Palvelin mallintaa analysaattorilaitetta standardissa määritellyssä muodossa. Se tarjoaa pääsyn laitteen tuottamiin tietoihin millä tahansa standardin mukaisella asiakasohjelmistolla. Prototyyppipalvelimen avulla voidaan demonstroida, miten yrityksen analysaattorilaitteet voidaan integroida Open Platform Communications Unified Architecture -standardiin.

Tämä opinnäytetyö auttaa yritystä arvioimaan Open Platform Communications Unified Architecture -standardin tarjoamia mahdollisuuksia tuotteilleen. Seuraavat kehitysvaiheet palvelinohjelmistolle olisi integroida se tiukemmin analysaattorin laitteiston kanssa ja tarjota enemmän analysaattorilaitteen ominaisuuksia asiakasohjelmille.

Avainsanat: OPC UA, Sulautetut järjestelmät, XRF

Contents

List of Abbreviations

API:	Application Programming Interface
DI:	Devices. OPC UA companion information model.
HHA-FI:	Hitachi High-Tech Analytical Science Finland Oy. A subsidiary of the global Hitachi High-Tech Analytical Science company.
MCU:	Microcontroller Unit. Compact computer in a single circuit.
OPC:	Open Platform Communications. Originally, the abbreviation came from Object linking and embedding for Process Control.
OPC UA:	Open Platform Communications Unified Architecture. Data exchange standard used in industrial automation space.
OS:	Operating System
PKI:	Public Key Infrastructure. A framework to manage public encryption keys.
RPC:	Remote Produce Call. Communication protocol.
SDK:	Software Development Kit. A collection of software development tools and libraries.
UA:	Unified Architecture. OPC UA term.
XRF:	X-Ray Fluorescence. A measurement technique used in the elemental analysis of materials.

1 Introduction

In today's fast-evolving technological landscape interoperability between different devices is becoming increasingly difficult. Because of the technical issues, many vendors prefer to only support communication between their own devices in their propriety ecosystems. Standardization is the key element to mitigate compatibility issues with software and hardware, especially in industrial applications. An example of an established standard is the Open Platform Communications Unified Architecture (OPC UA) standard, which is mainly used in industrial settings. The OPC UA is a communication standard, the key function of which is to provide a common interface for data exchange between devices.

This final year project was carried out for Hitachi High-Tech Analytical Science Finland Oy (HHA-FI), a subsidiary of the global Hitachi High-Tech Analytical Science company. HHA-FI specializes in analyzer devices and especially in X-Ray Fluorescence (XRF) handheld analyzers that are used to identify the composition of a wide variety of materials. These XRF analyzers have applications in many fields with different use cases. For instance, XRF based technology is utilized by the latest Mars exploration rovers to provide detailed analysis of the collected samples.

HHA-FI has recognized the increasing significance of software and its integration capabilities in their devices. HHA-FI has been exploring integration options for their analyzers and has chosen the OPC UA standard as the tool for the job. By providing a standardized way to access the device, the integration process of the device into an existing infrastructure becomes simpler and easier to achieve by 3rd party contractors. The goal of this final year project was to implement the OPC UA standard to HHA-FI's analyzer device. The OPC UA was chosen because of its established status in the industrial automation space. With the integration, HHA-FI's clients can use the OPC UA compliant client software of their choice to connect to the analyzer. The client connection provides access to the data produced by the device. The connection can be

used for further processing of the data and to connect the data to an OPC UA compliant automation system.

The thesis contains six sections. After the introduction, Section 2 provides the background information of the XRF analyzer targeted by the developed software. Section 3 discusses relevant key concepts of the OPC UA standard. Section 4 describes design decisions taken in the project and Section 5 analyzes the implementation phase. Finally, Section 6 contains results and suggestions for future development.

2 Background

This section provides background information on the core features of handheld XRF analyzers and their operating principles, with a due consideration for radiation safety. Additionally, it provides an overview of the HHA-FI XRF analyzer's software architecture.

2.1 XRF analyzers

Handheld devices with the ability to instantly read the elements of a solid, liquid, gas or plasma sample only existed in science fiction for a relatively long time. Only large laboratory devices used to be able to achieve that. [1, p. 1.] First portable devices with radioactive x-ray sources started to emerge in the seventies. These devices usually consisted of two parts, a handheld analyzer unit and a secondary unit containing a battery and other heavy components. At the beginning of the 21st century, technology had advanced to the point that it was possible to use a tube-based x-ray source with handheld XRF analyzers. [1, p. 6.] Figure 1 illustrates a tube based XRF analyzer's measurement principle.

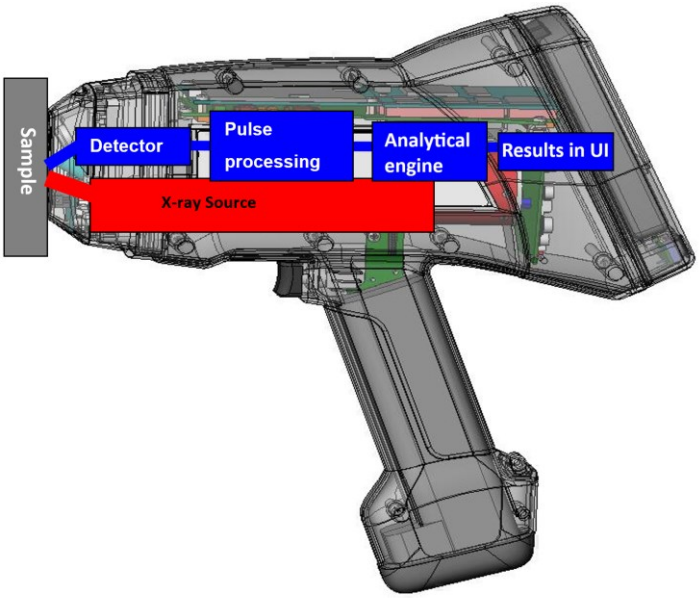

Figure 1. XRF analyzer's measurement principle [2, p. 1].

In the measurement cycle the analyzer sends x-rays towards the target sample as illustrated in Figure 1. During the process of the x-rays hitting the sample, photons are emitted to the detector. [1, p. 16.] The X-ray spectrum is produced from the pulses gathered by the detector and counted by a digital pulse processor. The composition of the sample is analyzed from the produced x-ray spectrum with an elemental analysis calculation software component. [1, pp. 19-21.] Finally, the result provided by the analysis calculation is displayed to the user. The result contains a list of elements that the analyzed material consists of.

Hitachi High-Tech Analytical Science employ this tube-based x-ray source solution in their X-MET range XRF analyzers [3]. Figure 2 illustrates an X-MET8000 analyzer.

Figure 2. X-MET8000 [3].

As seen in Figure 2, X-MET line is a typical example of a modern handheld XRF analyzer. It is equipped with a touch screen user interface and an easily removable battery. The analyzers have various use cases, usually in an industrial field. They can be used, for example, in a manufacturing process to provide quality control for the used material.

2.2 Radiation safety

XRF analyzers emit radiation during measurement. Although they operate at relatively low power, especially when compared with dental x-ray equipment, users should be trained in operating the device appropriately [1, p. 179]. Proper handling of the analyzer ensures that any potential leakage of radiation from the device is negligible. The radiation level from the device depends on chosen settings. Table 1 illustrates the worst-case scenarios for the radiation exposure from the device.

Table 1. Permissible radiation exposure [4, p. 9].

Location	Skin dose rate mSv/h	Time to permissible limit – skin[1]	Time to permissible limit – eye lens[2]
Main beam 45kV/40uA			
Sample window surface	2061	1min 37s	0.26s
100 cm distance	0.5	100h	30h

Examining Table 1 shows that even with directly exposed skin to the main beam radiation, there is still sufficient time for one to react and power off the device before reaching the maximum acceptable dosage limit of 50 mSv per year. However, eyes are notably more vulnerable to radiation, with a considerably smaller acceptable dosage limit. Although increasing the distance from the radiation source rapidly reduces the dosage, the effective dosage limit of 6 mSv per year can still be reached relatively quickly. [5.]

The user is quite safe from the radiation if the right tools are chosen, depending on the target sample, and the analyzer is operated properly. Figure 3 displays the correct way of operating the X-MET XRF analyzer.

Figure 3. Closed and open beam measurement [3; 6].

Figure 3 shows two options for a closed beam setup. The first entails shielding the user from radiation using a robust security stand, while the second option involves the use of a compact radiation shield cup for low-intensity measurements. The closed beam configuration becomes essential when measuring small samples incapable of effectively absorbing the radiation emitted by the device. On the contrary, an open beam measurement is suitable for larger targets with high radiation absorption capacity. It is important to note that during an open beam measurement, the user must refrain from touching the sample, and no body parts should be positioned behind the nose of the analyzer, as illustrated in Figure 3.

2.3 Software architecture of X-MET

The X-MET analyzers of HHA-FI are frequently employed in industrial environments. In such settings, the data generated by the device is usually needed in some further process, e.g., for controlling an automation process. This project aimed to streamline the integration between X-MET and the third party's own software by leveraging the well-established OPC UA standard.

The integration is achieved by adding an OPC UA server software component to the X-MET analyzer's existing software architecture. Figure 4 illustrates the analyzer's software architecture.

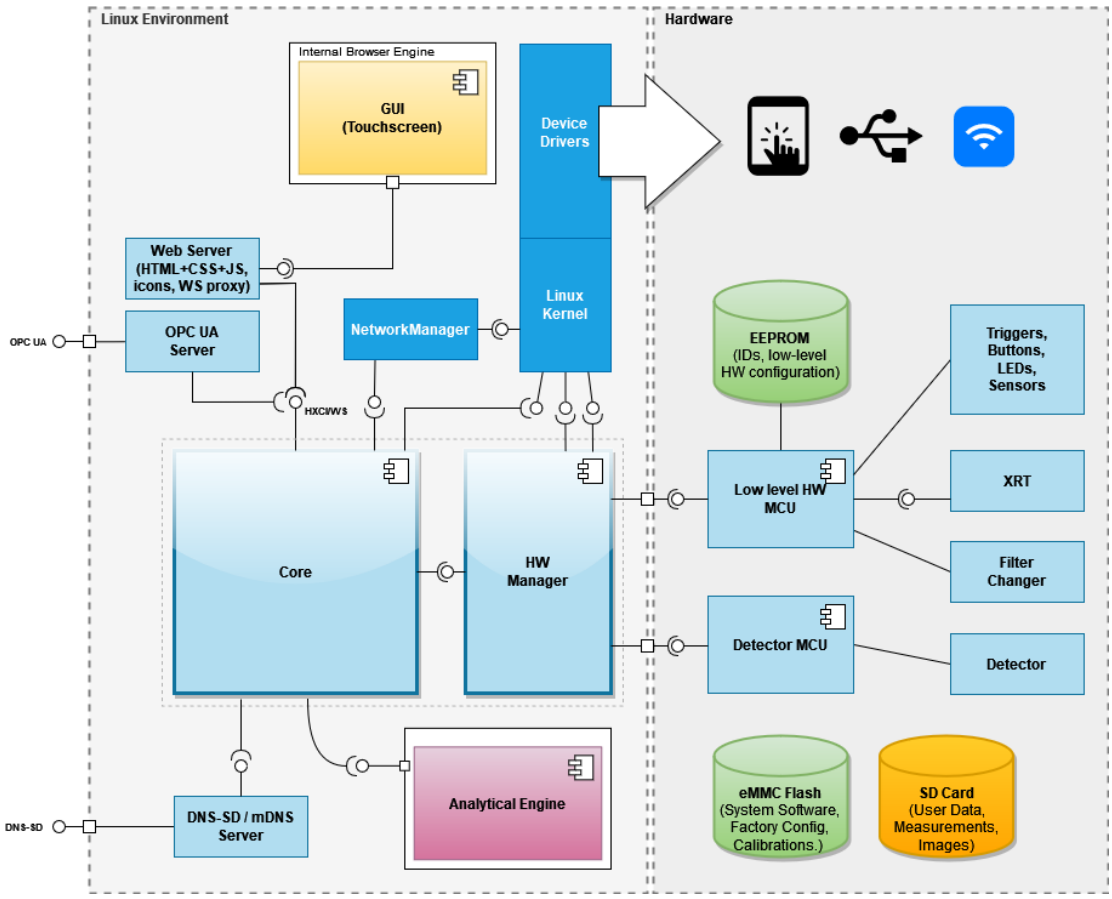

Figure 4. X-MET analyzer's software architecture [7].

Figure 4 describes the architecture with a division into two logical segments, a primary module operating in a Linux environment and the hardware segment governed by two microcontroller units (MCU). In the hardware segment, Detector MCU is responsible for using the Detector to get raw measurement results. The low level HW MCU operates the rest of the onboard components, such as buttons, LEDs and EEPROM memory. The HW Manager software component is used to control these two MCUs.

In the Linux environment segment, the Core is the most important component, controlling the device by leveraging other components such as HW Manager. For example, the Core handles data acquisition from onboard sensors via the HW Manager and uses the Analytical Engine to process the raw data into measurement results.

The new OPC UA server component will be running in the Linux environment. It communicates with the Core via an internal web socket connection, employing an inhouse remote produce call (RPC) protocol, elaborated in Section 5. With the connection to the Core, the server gains access to the rest of the hardware of the device. For instance, the OPC UA server can initiate a measurement by sending a request to the Core, which, uses the HW Manager to start the measurement. After the measurement is complete, the Core receives the raw data and employs the Analytical Engine to generate a measurement result and communicates the result back to the OPC UA server. The OPC UA server is explored more in detail in Section 4 and Section 5.

3 OPC UA key concepts

This section describes the key concepts of the OPC UA standard. OPC UA is a platform independent standard managed and developed by the OPC Foundation. It allows the exchange of data between devices in a standardized way. It is mainly used in industrial automation. [8.] The first sub-section explores the origin of the standard. Later sub-sections focus on nodes, overall architecture, and the security aspects of the OPC UA standard.

3.1 History

In its initial release in 1996, the OPC UA standard was known as the OPC standard. Originally, the acronym OPC originated from object linking and embedding for process control. Presently, this acronym has evolved to signify Open Platform Communications (OPC). [8.] As industrial automation systems started to embrace software-based control in the early nineties, the need for standardized communication between devices started to emerge. The industry was in need of a standardized way to access the data generated by various devices with several bus systems, interfaces and protocols, which led to the development of the OPC standard. [9, p. 1.]

The first release of the standard consisted of an OPC Data Access specification, the purpose of which was to act as an interface providing standardized access to data through the drivers of the device. [9, p. 1.] With this abstraction it was possible to use the generic read/write requests without writing device-specific code. The generic requests are converted to device-specific requests by the Data Access specification. [8.] The success and quick adaption of the standard was partly possible because it relied on existing Windows technologies such as Component Object Model and Distributed Component Object Model. [9, p. 1.] Later in addition to the Data Access specification more specifications were added to the OPC standard [10]. The additions were OPC Alarms & Events and OPC Historical Data Access specifications. Nowadays these older Windows-dependent specifications are known as OPC Classic [8].

The OPC foundation released the OPC UA standard in 2008. As the name Unified Architecture suggests, it integrates all the previous OPC Classic specifications into one unified architecture. The OPC UA standard adds new features compared with the OPC classic. The new Unified Architecture (UA) features are listed below:

- **Discovery:** find available servers in the network.
- **Address space:** all data is represented with file-folder structure.
- **On-demand:** access permissions to read and write data.
- **Subscriptions:** monitor data changes.
- **Events:** notify on important information.
- **Methods:** clients can execute functions defined on the server.

Additionally, UA adds architectural improvements to the standard. It is not dependent on any operating system (OS) and has focus on the scalability. The scalability is achieved with multi-layered design and extendable information models. Information models are discussed later in this section. Backwards compatibility between Classic and UA standards is supported with wrappers. [11.]

3.2 Node

One of the fundamental concepts in the OPC UA is a node. Nodes represent individual points or entities in the system. A collection of nodes is called namespace. Namespaces are studied more in detail later. A node always has a class, and there are eight different classes: object, variable, method, view, object type, variable type, reference type and data type. [9, p. 22.] Figure 5 displays all the available node classes in the OPC UA.

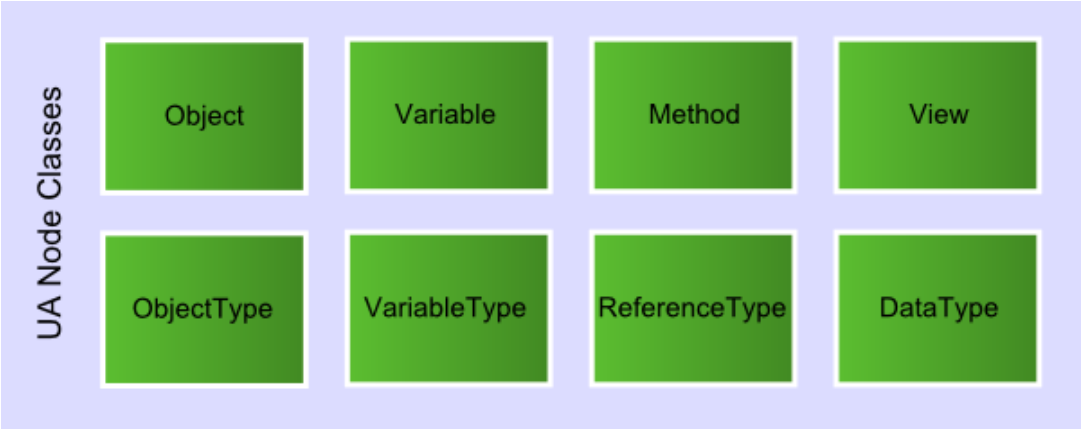

Figure 5. OPC UA node classes [12].

The most important classes seen in Figure 5 are variable, method and object classes. Variables are used to represent values, usually a sensor's measurement value. Methods can return a result and can be called by the server or the client software. Objects can be used to follow the object-oriented programming design philosophy. They collect nodes to logical groups. [9, p. 30.]

A node consists of attributes and references. Figure 6 illustrates how nodes are tied together with reference to each other.

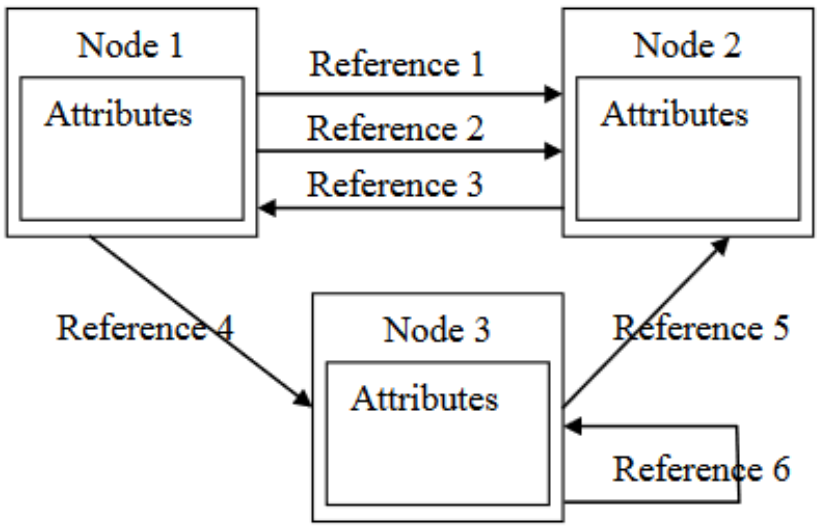

Figure 6. Nodes and references [9, p. 22].

Nodes are defined by their attributes. A node can have a different set of attributes, and they are determined by the node's class. Despite the class, all nodes have some common base attributes. For example, an ID attribute is mandatory for every node and it is used to reference the node in the server. [9, p. 22.] Figure 6 shows that nodes can have multiple references between each other. Information models are used to define a set of nodes and their relationships with each other. The concept of the information models is explained in the next sub-section.

3.3 Architecture

The architecture of the OPC UA consists of several semi-independent modules. The standard gathers these modules together to provide a unified architecture. [13.] Figure 7 below illustrates the architecture.

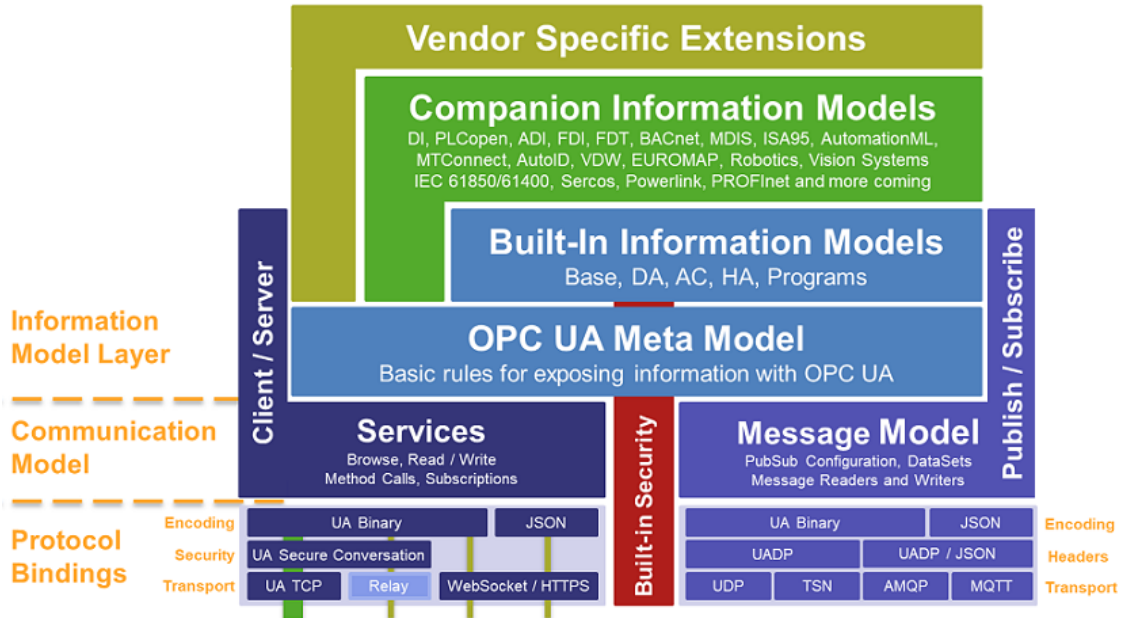

Figure 7. OPC UA architecture [13].

As illustrated in Figure 7, the modules can be organized into three categories: information model layer, communication model and protocol bindings. The information model layer category consists of several information models. The information model is a set of nodes connected to each other with references, in

other words a namespace. The meta model defines base concepts and rules for the information modelling concept. All the information models depend on the meta model. Built-in information models provide basic OPC functionality, such as data access. Companion information models define use case specific models. For example, the ADI specification describes an information model for analyzer devices. It depends on a generic device specification, which is defined in the DI companion model. Vendor specific information models are used to define a vendor's own products. The OPC foundation recommends that vendors use companion information models for their products as using a standardized companion information model to model a physical device ensures that all the same type of devices have a similar namespace hierarchy. If a companion model is insufficient, a vendor can extend it with a vendor specific extension. This extension is a vendor specific information model that depends on an existing companion model. [13.]

The communication model category consists of services and the message model. The OPC UA services are used to implement interaction between OPC clients and servers. The services are abstract as they define information exchange between OPC UA applications. The message model is responsible for the publish / subscribe feature of the OPC UA. Concrete protocol implementations of the services and the message model are defined in the protocol bindings category. [13.]

3.4 Security

The OPC UA standard can seamlessly run across various operating systems and platforms, spanning from Windows desktops to embedded devices. Given this scalability, security measures must also adapt accordingly. OPC UA addresses this challenge by defining security policies, which are a set of security components. As OPC UA applications can function as servers, clients, or both, the primary security emphasis lies in ensuring the secure transport of data between these applications. This security framework operates across three distinct layers illustrated in Figure 8. [14.]

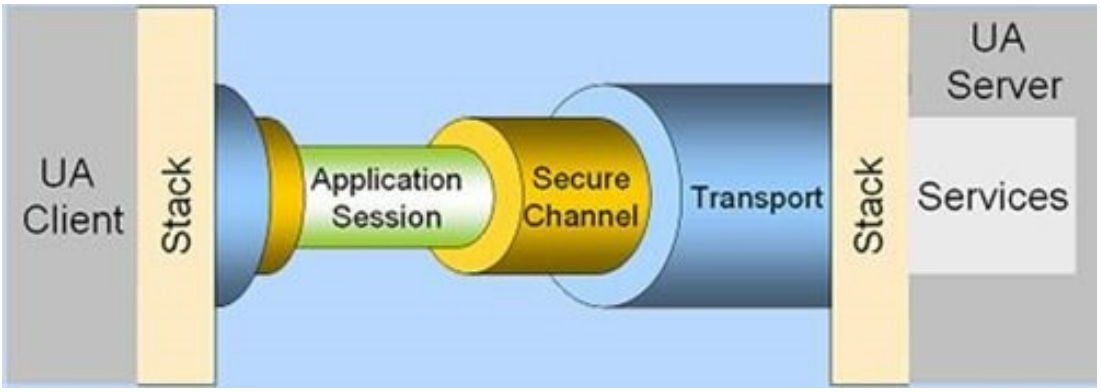

Figure 8. OPC UA security layers [15].

Figure 8 illustrates the application session, the secure channel, and the transport layers. The application session can transmit, for example, measurement data between a client and a server. The application session runs on top of the secure channel, which is used to exchange certificates and manage the data integrity with signing and encryption. The application session and secure channel layers rely on certificates for the security. The OPC UA standard does not specify infrastructure related to creating, storing, and managing the certificates. The final layer, the transport layer, oversees the socket connection between OPC UA applications, focusing primally on things like IP addresses and ports. [14.]

The connection establishment between a server and a client involves a four-step process. Initially, the client starts communication with the server to inquire about available endpoints for the connection, which is carried out unencrypted. Then the client selects a specific endpoint for the connection and assesses the trustworthiness of the server's certificate. Following this, an 'openSecureChannel' request is transmitted, accompanied by a selection of security options such as 'none,' 'sign,' or 'sign&encrypt'. The server, in turn, retrieves the client's certificate from the unencrypted portion of the message and, upon validation, decrypts the data using tools associated with the chosen endpoint. The third step involves the creation of a session atop the secure channel. Finally, the session is activated, requiring the input of a username and a password. This process is illustrated in Figure 9. [9, pp. 209-215.]

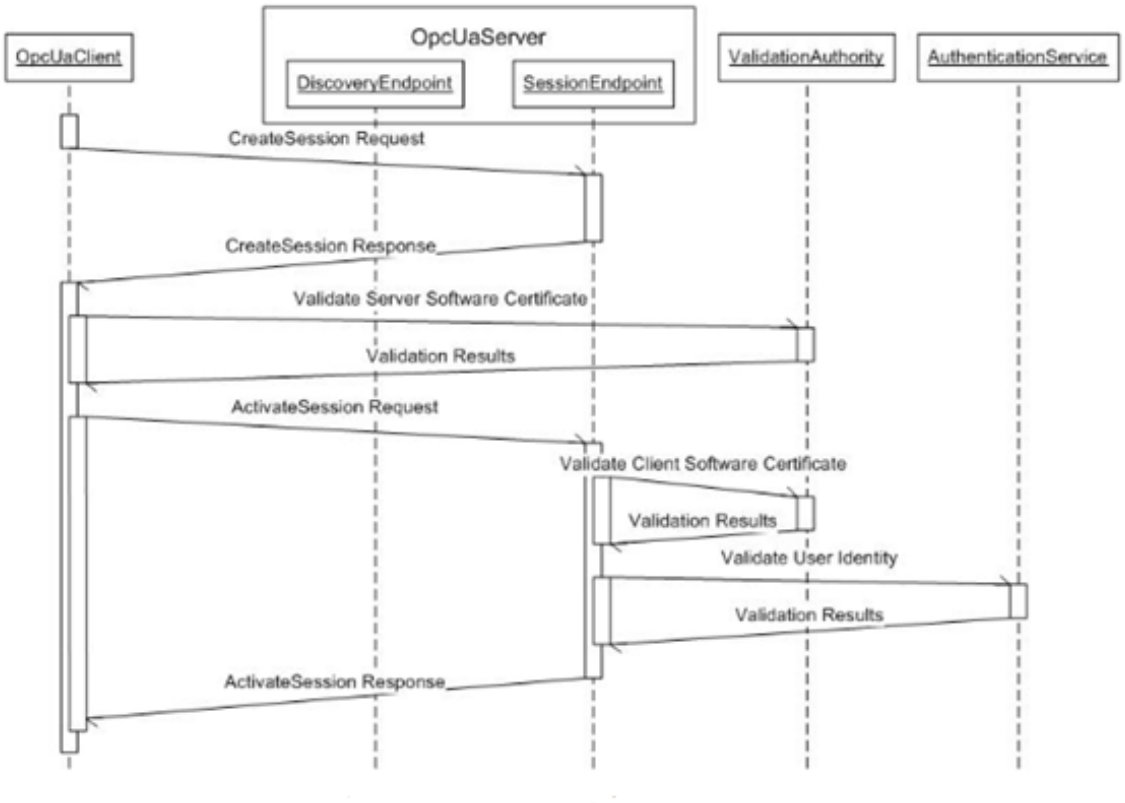

Figure 9. Establishing connection [9, p. 215].

Figure 9 shows that a quite bit of validation is needed before connection is established between two OPC UA applications. Both symmetric and asymmetric keys can be used for the validation. Using the asymmetric model has an advantage as a public key can be shared between applications without worry. With symmetric keys, the private keys must be transferred securely between the applications before they can communicate with each other. This transfer is not defined in the OPC UA standard. [14.]

OPC UA relies on X509 certificates for signing and encryption. Signing is used to validate that a received message is from an expected sender. This ensures that messages from rogue sources are ignored. Encryption ensures that messages cannot be read without decryption. The used encryption protocol method depends on the chosen OPC UA security policy. The Public Key Infrastructure (PKI) framework defines the standards for the X509 certificates

and outlines procedures for their management. Tools such as ‘openSSL’, ‘openxpi’, and Windows PKI can be employed for efficient PKI management. [14.]

4 Project design

The purpose of this project was to integrate an OPC UA server into the X-MET analyzer device. The project had two planned stages. The OPC UA server is first implemented as a standalone application in the Linux and Windows environments. At this point its only functionality is to start the server and publish sample data to the network. Published information can be monitored with any OPC UA client software; there are many to choose from. This first stage of the project is essential to set up the work environment and workflows. After the desktop application is up and running, the development of an embedded application can begin. The embedded application is a service for the analyzer's embedded Linux environment. At this stage the first objective is to port the desktop application to the embedded environment. After that, a connection from the application to the rest of the device is implemented to enable the extraction of real data from the analyzer. Section 2 explored in detail how the new OPC UA server application is connected to the rest of the analyzer's software architecture.

This section contains descriptions of the design choices made for this project. The first sub-section explores the available OPC UA SDKs and the one that was chosen. After that, a design of the desktop and embedded servers are explained. Later, the architecture of the developed server application is described, and the testing procedure made for the project is discussed.

4.1 OPC UA SDKs

Many OPC UA software development kits (SDK) are available, using both proprietary and open-source licences. The OPC UA foundation maintains its own SDK, which is accessible for the public for free. Figure 10 describes the contents of an OPC UA SDK.

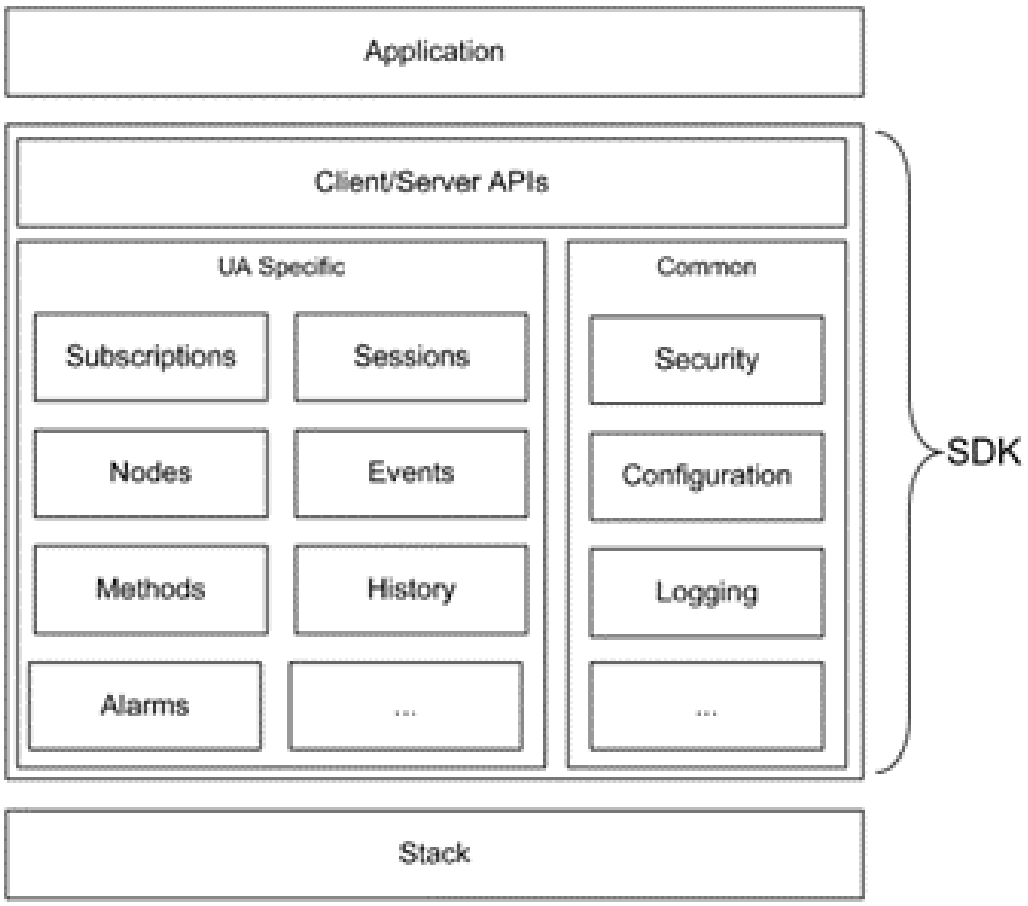

Figure 10. Structure of an OPC UA SDK

Figure 10 illustrates that the stack is not part of an SDK. The stack contains implementations of low level OPC UA specifications. Both ANSI-C-based and a C#-based stack is provided by the OPC foundation. Third party SDKs can leverage the provided stacks. Typically, an SDK implements high level OPC UA concepts and provides tools that are not defined in the standard. The SDK provides client and server application programming interfaces (API), so that the OPC UA application using the SDK can access its implementations. [9, pp. 261-263.]

For this project open62541 SDK is selected alongside with the open62541pp C++ wrapper. Open62541 is an implementation of the UA Stack, provided by the OPC UA Foundation and is made with C99. The developed OPC UA server application will access open62541 functions through the open62541pp wrapper in this project. Open62541 SDK was selected because it has an appropriate

open source license, and it is rather mature product itself. The existing C++ wrapper is used to focus on application development rather than on the infrastructure of the application.

4.2 Desktop application

The initial phase of the project involves establishing a standalone OPC UA server application. The first step is setting up the development environment with the chosen programming language for the project, C/C++. The server leverages the open62541 open-source OPC UA implementation, made with C99. The Conan package manager is used to manage dependencies, the open62541 library is acquired via Conan, and the server is built using the ‘conan build’ command. The OPC UA is platform independent, so the desktop server application should run in any desktop OS. Windows is used in the development. The Linux build is tested with an Ubuntu distribution.

The creation of a custom OPC UA information model, called the vendor specific information model in OPC terms, is initiated early in the project. Information models and vendor specific extensions were explained in Section 3. This new information model is used to model the X-MET analyzer within the OPC UA framework. The desktop application uses this information model to display sample data to a client application. The sample data simulates the data gathered by the analyzer. Client software is essential to test the connection to the server. OPC UA server should seamlessly work with any client software adhering to the OPC UA standard. For this project the UA Expert client was chosen for testing because of its features and graphic user interface. Figure 11 illustrates a client connected to the server.

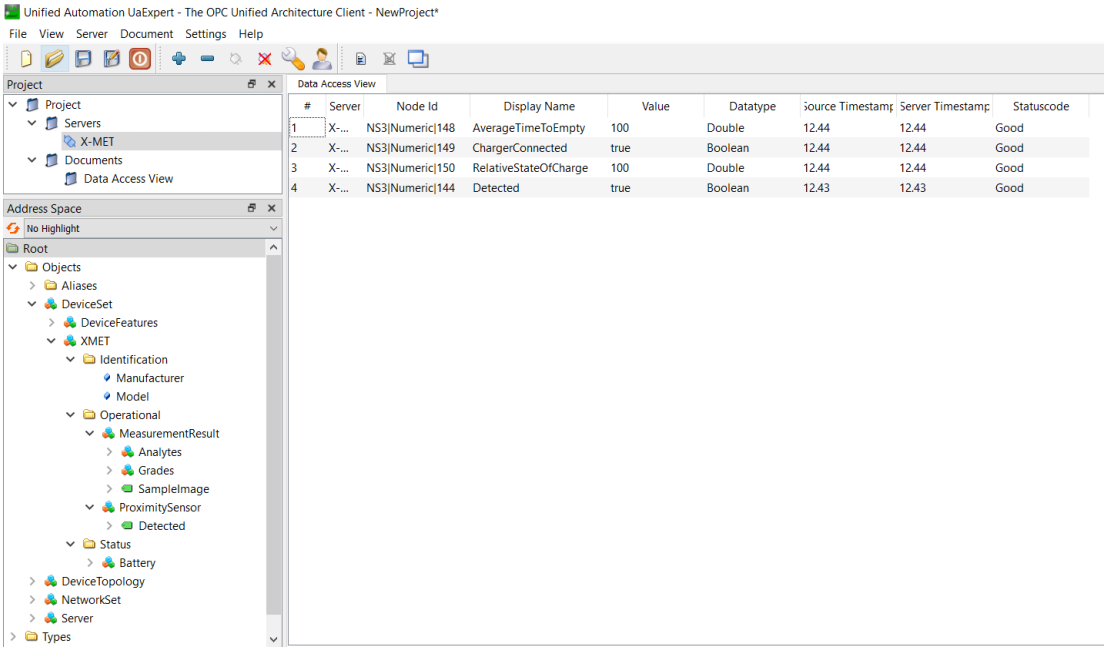

Figure 11. OPC UA client program connected to the desktop server

It can be seen in Figure 11 that when a client connects to the desktop server it shows the X-MET analyzer object that represents the device and its characteristics in a hierarchical tree view. The client can monitor the analyzer's sensor values, such as checking the battery level of the device in the data access view. In the embedded server application, these values come from the device, but in the desktop application these values are simulated and can be set manually.

4.3 Embedded application

The software of the X-MET analyzer devices relies on 'OpenEmbedded', a build framework for embedded Linux, allowing developers to craft Linux distributions tailored for embedded systems. The integration of the OPC UA server into the existing Linux environment is accomplished through the addition of a Yocto layer of the server. Because of the OPC UA standard's platform-independence, the code designed for the desktop server application will work on the embedded device without modifications. Only build tools need to be changed to Bitbake to build the embedded server application into the embedded Linux environment.

At this point, the next step for the project involves establishing a connection between the server application and the X-MET device. The analyzer features a core service that enables the connectivity with the server via the WebSocket client. The server application code has been C code to this point. To enable the WebSocket connection, porting the existing server code to C++ is necessary, as the WebSocket library provided by the X-MET analyzer is a C++ library alongside with other necessary libraries. As the focus of this project is on application development, a ready-made wrapper is used rather than developing it from scratch. The switch from C to C++ is achieved with open62541pp, a C++ wrapper made for the open62541 SDK.

4.4 Application architecture

The OPC UA server application operates within an embedded Linux environment, leveraging the open source open62541 OPC UA implementation. Given that open62541 is developed with C99, the adoption of the open62541pp C++ wrapper adds a layer of convenience. While not all features of open62541 are currently implemented in the open62541pp, the server application can use the C API of the ope62541 for any missing functionality. The application architecture is illustrated in Figure 12.

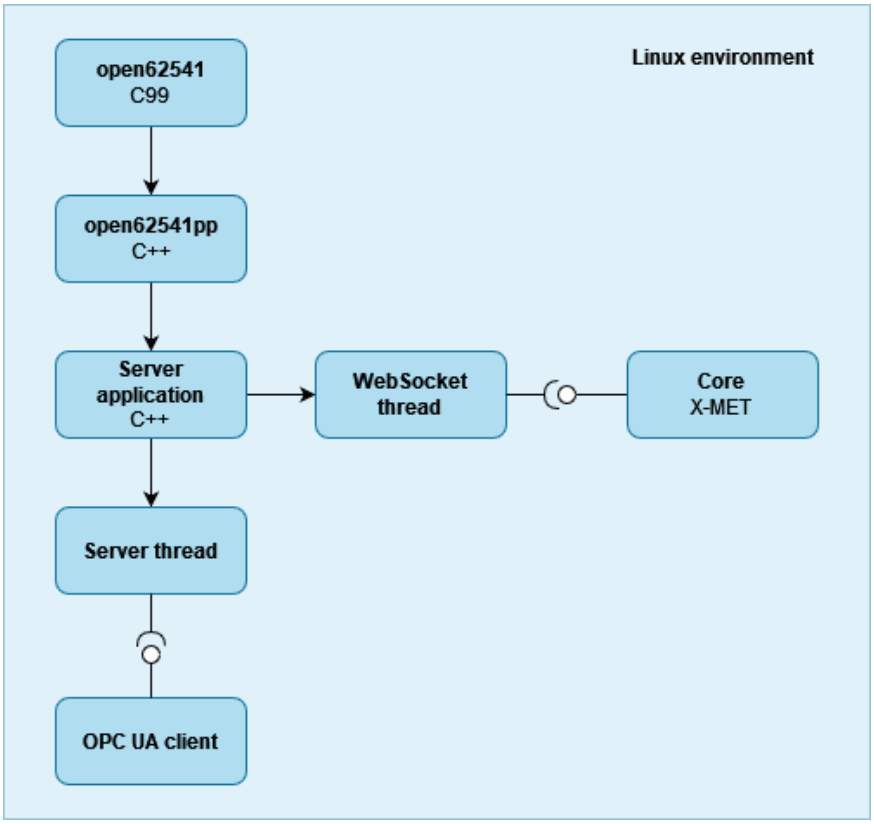

Figure 12. Server application architecture

As seen in Figure 12, the server application depends on the open62541 and open62541pp libraries. It has two running threads. The server thread manages the server's connection to the client, serving the client's requests. The server thread is implemented by using the open62541 server API, accessing it via the open62541 wrapper. The WebSocket thread manages the internal WebSocket client connection to the Core service. The connection to the Core enables access to the X-MET analyzer. This connection to the X-MET is based on sending requests and listening for event and uses custom RPC protocol for the communication. The WebSocket thread listens out for events coming from the Core service and acts accordingly depending on the received event. For example, if an event about battery level change is received, it updates the corresponding OPC UA node's value.

4.5 Testing

The open-source automation server Jenkins is used to execute automatic build tests. The tests are run if changes are detected in any of the repository's branches. Changes are scanned once a day. Figure 13 illustrates the result of a build test.

Figure 13. Jenkins build test result

It can be seen in Figure 13 that the test consists of several stages. The build stage is the most important. It is used to check that the build is successful and to provide the build artefacts available for download. The analysis stage runs the static code analysis 'Cppcheck' tool to provide guidance for cleaner code. The publish stage is used to provide an executable file, available for download in the Jenkins interface.

5 Project implementation

This section describes in detail the implementation part of the project. It explores the vendor specific extension made for the server application. It explains the details of the in-house RPC protocol used for communication with the hardware of the X-MET analyzer. The main loop of the application is described, and the security aspects of the project considered.

5.1 X-MET information model

One of the key concepts of the OPC UA is information modelling. To represent the X-MET device in OPC UA context, it must be modelled by using nodes that are connected to each other with references. Open62541 has two ways to populate the OPC UA server's namespace with nodes. One solution is to manually write code to define and add nodes to the server. The other, more versatile solution is to use a 'NodeSet2.xml' file. The format of the file is defined by the OPC Foundation. The open62541 library includes a tool called XML Nodeset Compiler. The tool can be used with a 'NodeSet2.xml' file to generate C code containing the information model. This solution is used in this project.

A 'NodeSet2.xml' file can be written manually but it is prone to syntax errors. A better solution is to first write a 'Model.xml' file, then transform it to the NodeSet2.xml format. Figure 14 and Figure 15 show differences between the 'NodeSet2' and 'Model' formats.

```
144     <!-- XmetObject -->
145     <Object SymbolicName="XMET:XMET" TypeDefinition="XMET:XmetType" SupportsEvents="true">
146         <Description>Object instance of the Xmet Type.</Description>
147         <References>
148             <Reference IsInverse="true">
149                 <ReferenceType>OPCUA:Organizes</ReferenceType>
150                 <TargetId>DI:DeviceSet</TargetId>
151             </Reference>
152         </References>
153     </Object>
```

Figure 14. Model.xml

```
<UAObject NodeId="ns=1;i=99" BrowseName="1:XMET" EventNotifier="1">
  <DisplayName>XMET</DisplayName>
  <Description>Object instance of the XmetType.</Description>
  <References>
    <Reference ReferenceType="HasComponent">ns=1;i=104</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=134</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=145</Reference>
    <Reference ReferenceType="Organizes" IsForward="false">ns=2;i=5001</Reference>
    <Reference ReferenceType="HasTypeDefinition">ns=1;i=47</Reference>
  </References>
</UAObject>
```

Figure 15. NodeSet2.xml

As one can be seen in Figure 14 and Figure 15, the NodeSet2 format is more complex and requires the use of numeric IDs for namespace and identifier indexes. On the contrary, the ‘Model.xml’ model format is far simpler. The OPC Foundation provides a UA-ModelCompiler tool to transform the ‘Model.xml’ file to ‘NodeSet2.xml’ file. The foundation does not provide a specification for the model files. A namespace is generated with the following process. First, ‘UA-ModelCompiler’ is used to generate a ‘NodeSet2.xml’ file. Then, the open62541’s Nodeset compiler can be used to generate the source files. In the server application code, the namespace can be loaded with a single line command.

The X-MET information model is constructed by following the OPC UA specifications. All information models have dependency on the base OPC UA information model, which defines all the basic functionality of an information model. The X-MET model is a vendor specific extension to the Devices (DI) companion information model provided by the OPC foundation. DI specification ensures that devices are represented in a standardized way in the OPC UA. Following the DI specification has other benefits as well. Some OPC UA client software have special support for the DI specification, making it easier to manage multiple devices in the client. The developed OPC UA server application supports this too, as the X-MET information model follows the DI specification. Figure 16 illustrates the X-MET information model.

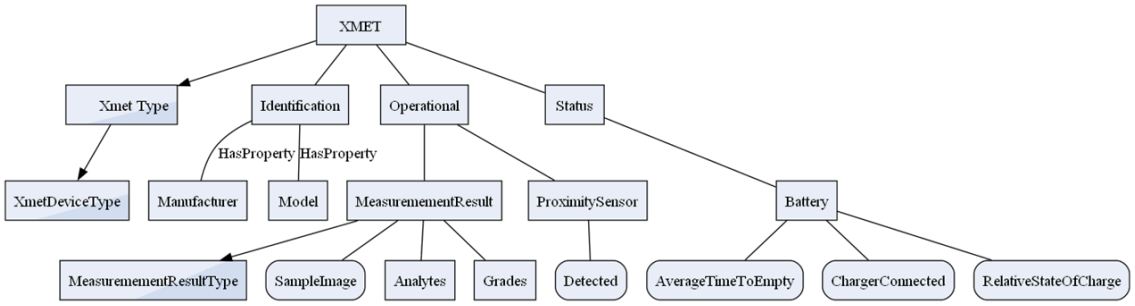

Figure 16. X-MET information model

As seen in Figure 16, the X-MET object is an instance of ‘Xmet Type’, which is of the ‘XmetDeviceType’ type. ‘XmetDeviceType’ is an abstract base type for all X-MET instruments. It provides an identification object that has Manufacturer and Model properties. These properties along with the identification object are standardized by the DI companion specification that the X-MET model depends on. Objects cannot be instanced from abstract types, so ‘Xmet Type’ is needed. It provides ‘Operational’ and ‘Status’ objects. The ‘Operational’ object collects parameters and methods useful during normal operation. ‘Status’ collects parameters which describe the health of the device, such as diagnostic, describing parameters. Both ‘Operational’ and ‘Status’ parent objects are standardized by the DI.

The children of the ‘MeasurementResult’ object are dynamically constructed during runtime. The OPC UA server application establishes a connection to the Core software component via a WebSocket client. Upon the completion of a new measurement, the server requests results from the Core and creates new nodes to the ‘MeasurementResut’ based on the result. ‘MeasurementResult’ children are constructed based on values in the response of the Core. ‘MeasurementResult’ always has analytes and grades objects as children. Their children vary based on the measurement result. The server only holds the latest measurement result. When it receives information about a new measurement, it uses the destructor to delete all children under the ‘MeasurementResult’, and calls the constructor to create them again based on the new response values.

5.2 In-house RPC

The in-house RPC protocol is a remote procedure call interface specification and protocol for inter-process communication. It is used to abstract away low-level hardware communication in the X-MET analyzer. It uses the streamlined MessagePack RPC specification. The RPC is designed to be compatible with any transport layer protocol. The current implementations uses WebSocket as the transport layer.

RPC communication is based on sending requests and receiving responses. The response contains either a result or an error message to the request. A third message type is Event. Events are used to push notifications to clients. Requests consist of a resource path ID and the name of the method that is to be called. The response provides a value or an error to the request made. If a request is made, the server must reply with a respond message. Events may be sent to the client as they occur in the server. The source of the event can also be identified, such as the resource which generated the event. When an error occurs the response message must contain an error field that contains a list of errors.

5.3 Repository layout

The project consists of two repositories, ‘opc-ua-server’ and ‘meta-opc-ua’. The main repository, ‘opc-ua-server’ is constructed with a pitchfork layout illustrated in Figure 17.

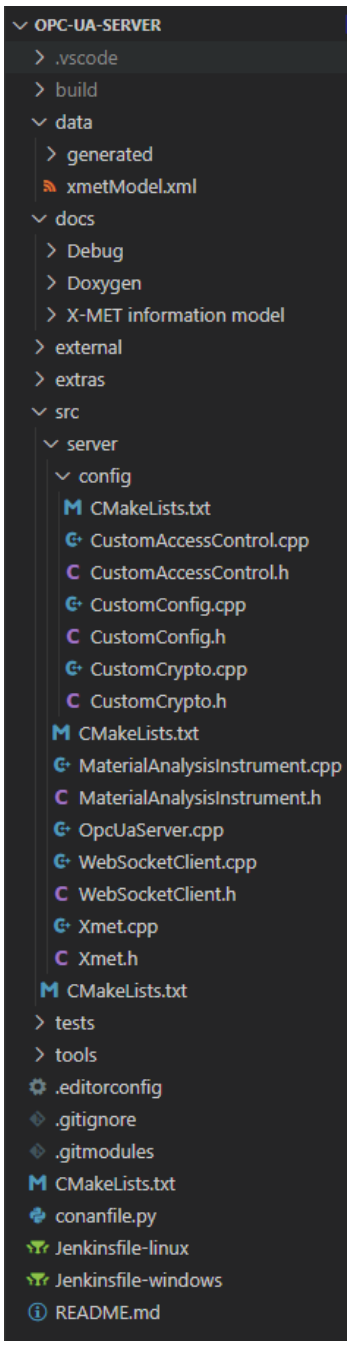

Figure 17. Repository's layout.

Figure 17 showcases the pitchfork layout. Build artifacts are stored in a build directory, which is not part of the remote repository. All the source files are inside the src folder. The data folder contains non-source files, such as INI files. Development tools are stored in a tools folder. Test cases, documents and examples go to corresponding test, doc, and example folders. The extras folder

contains optional submodules and ‘external’ folder source code for mandatory dependencies.

The source code is organized into 'src/config' and 'src' folders. 'src/config' contains essential functions and classes dedicated to configuring custom settings for the server, including aspects such as access control and encryption. Located inside ‘src’ folder the analyzer class is an abstract class, designed for enabling future integration with various analyzers. The X-MET class inherits the analyzer class. It models the X-MET device by loading nodes from the X-MET information model. It is responsible for managing these nodes. For example, it provides functions to update the specific object node’s values. ‘OpcUaServer.cpp’ initializes and launches the WebSocket and OPC UA server threads.

The ‘meta-opc-ua’ repository is used to add the server as a layer to the X-MET analyzer’s Linux environment. It uses the source code from the ‘opc-ua-server’ repository. Figure 18 illustrates the simple layout.

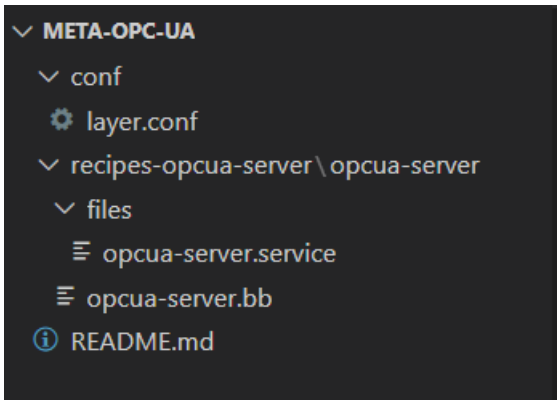

Figure 18. ‘meta-opc-ua’ layout.

As can be seen in Figure 18, ‘meta-opc-ua’ contains only embedded Linux specific configuration and build files. CMake and source files from the ‘opc-ua-server’ repository are used to build the project.

5.4 Application logic

The server application has two loops running in separate threads. The WebSocket client connection is run in its own thread. It is used to connect to the X-MET's Core service via the in-house RPC protocol. This allows the server to communicate with the X-MET analyzer's hardware. The other loop is the server's main loop, used to manage the connection to the OPC UA client application. Figure 19 illustrates the main loop.

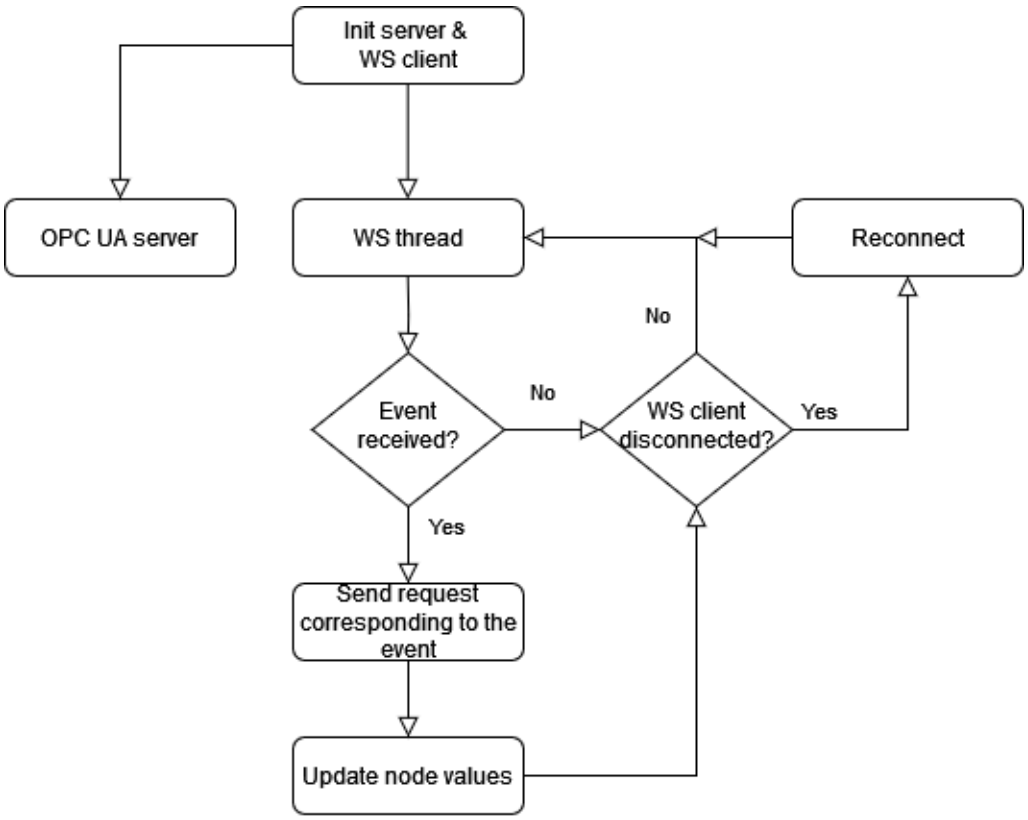

Figure 19. Server application's operation logic

As seen in Figure 19, first a WebSocket connection to the Core is created and the OPC UA server initialised and started. After the initialization a WS thread and an OPC UA server thread are started. If an event is received from the Core service, it is processed. If the event contains data belonging to some of the nodes, the received value is written to the corresponding node. In the event of RPC disconnection, the thread tries to reconnect. The OPC UA server handles communication between clients and the server.

5.5 Security concerns

The OPC UA has built-in security methods, for example access control to manage user's access to the server. A user can be limited to view only specific nodes or allowed only to read the node values without write access. Traffic between the server and the client can be encrypted with 'OpenSSL' or 'Mbed TLS'. The developed server application implements OPC UA security policies, 'none' and 'aes128'. 'None' is needed to announce available endpoints to the clients. Still, 'none' is not provided as a valid endpoint for the client to connect to. The only endpoint available for clients is the 'aes128' one. Clients need a username and password to establish connection. Certification files can be used instead of a password. The server creates its own certification file at the start-up. It also maintains a trust list, so only clients with trusted certificates can connect the server.

Adding the OPC UA server to the Linux environment of the X-MET analyzer opens a new attack vector for unauthorized access to the device. Because the server has access to the device's hardware, the radiation safety of the analyzer is threatened as well. However, this risk is mitigated by the fact that the X-MET device with the OPC UA server running will probably be used with the device installed on a fixed location. The installation can be done in a way that the operator cannot be exposed to the radiation, even if the device is accidently on. The built-in security of the OPC UA standard can be trusted as the standard is quite a mature product. However, a human error could lead to the failure of the server's security and grant the attacker access to the analyzer. The risk can be minimized by allowing the remote connection to the OPC UA server only from a local network. With the access to the OPC UA server, the attacker could execute several malicious acts such as deleting the measurement database from the device or starting an unscheduled measurement with the analyzer. This could lead to a situation where operators are exposed to radiation if the analyzer is measuring unexpectedly.

6 Conclusion

This section summarizes the results of the project and draws conclusions based on them. Additionally, suggestions for future development are given.

XRF analyzers are used to determine elemental composition of a wide variety of materials. The analyzation process emits harmful x-rays, so a user needs to be properly trained to use an XRF analyzer. HHA-FI produces handheld X-MET XRF analyzers.

The OPC UA standard is a communication standard, which is widely used in industrial automation. It allows devices from different vendors to communicate with each other. OPC UA applications are either servers or clients. A server application contains data structured into namespaces. A Namespace consists of nodes. A node can for example hold temperature data produced by a temperature sensor. The OPC UA client application can connect to a server and read and write data in it.

The purpose of this project was to implement an OPC UA server into the Linux environment of the X-MET XRF analyzer. To do that, the X-MET analyzer is modelled in the X-MET namespace information model, which follows OPC UA definitions for a vendor specific namespace extension. The implemented OPC UA server application uses the internal WebSocket connection of the X-MET to get sensor readings from the analyzer. The OPC UA client can be used to connect to the server and read the data produced by the X-MET.

6.1 Results

The X-MET analyzer’s embedded Linux environment now contains an OPC UA server. It can communicate with the analyzer using a WebSocket connection with the in-house RPC protocol. Any OPC UA compliant client can be used to connect to the analyzer to get access to the data produced by the device.

The X-MET information model was created because of this project. The information model is in the ‘Model.xml’ format that can be converted to the ‘NodeSet2.xml’ format by tools provided by the OPC UA foundation. The ‘NodeSet2.xml’ file can be used with a wide variety of OPC UA SDKs to generate the information model source code for the specific SDK in use. This ensures that the information is portable and not tied to any specific OPC UA SDK. It can also be easily expanded. After any modification the information model code can just be generated again without refactoring existing code.

Achieving the goal of this project, the X-MET XRF analyzer now has a standardized way for the client to connect and extract data. This is achieved with the OPC UA server that runs in the device’s Linux environment. Previously data access was achieved by REST style API connection. This requires knowledge of the X-MET device to integrate it into the client’s systems. The new OPC UA server integration ensures that the connection to the X-MET is standardized and any contractor familiar with OPC UA can integrate the device into the client’s system without any previous knowledge of the analyzer itself.

6.2 Future development

The scope of the server is limited to one target device, but the server could be developed further to provide a generic OPC UA server for multiple models of X-MET XRF analyzers.

The OPC UA server can be developed further by implementing more of the X-MET’s features to the server. The OPC UA allows clients to use methods to execute actions on the server side. This is an essential feature that should be implemented in the future on the produced server. For example, a method for starting the measurement could exist on the server, so the client could initiate the start of the measurement remotely. There could also be a method to search the device’s measurement database for old measurements and return their contents to the client.

The OPC UA foundation is planning on releasing a new Laboratory and Analytical Device Standard (LADS) Q1/2024. LADS will be implemented as an OPC UA companion specification. When the new specification is released, it will be used in the X-MET's OPC UA server. Both the X-MET namespace and LADS depend on the DI specification, so implementing this new standard in the existing custom X-MET namespace should be straightforward. This new specification is an excellent fit for the X-MET device.

References

1 Drake BL, MacDonald BL, editors. Advances in Portable X-ray Fluorescence Spectrometry: Instrumentation, Application and Interpretation. 1st edition. Cambridge: Royal Society of Chemistry; 2022. 548 pages

2 XRF presentation OIA. 2010. Internal documentation of HHA-FI.

3 Handheld XRF Analyzers | X-MET8000 Range [Internet]. Hitachi High Tech Analytical Science. [cited 2023 May 27]. Available at: https://hha.hitachi-hightech.com/en/product-range/products/handheld-xrf-libs-analyzers/handheld-xrf-analyzers

4 Lindholm S. 2020. HS 518 Radiation Safety Training. Internal documentation of HHA-FI.

5 Radiation and Nuclear Safety Authority (STUK). Säteilyaltistuksen enimmäisarvojen soveltaminen ja säteilyannoksen laskemisperusteet, 8.8.2014. 2014 [cited 2023 October 9]. Available at: https://www.stuklex.fi/fi/haku/ohje/ST7-2

6 Systems TMQC. X-MET 8000 Smart XRF Spectrometer [Internet]. Troy-Met Quality Control Systems. [cited 2023 October 24]. Available from: https://www.troy-met.com/en/product/191/x-met-8000-smart-xrf-spectrometer

7 Aho J. 2023. X-MET Software Architecture. Internal documentation of HHA-FI.

8 What Is OPC? [Internet]. OPC Foundation. [cited 2023 May 21]. Available from: https://opcfoundation.org/about/what-is-opc/

9 Mahnke W, Leitner SH, Damm M. OPC Unified Architecture [Internet]. 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009 [cited 2023 May 28]. Available at: http://link.springer.com/10.1007/978-3-540-68899-0

10 History [Internet]. OPC Foundation. [cited 2023 May 28]. Available at: https://opcfoundation.org/about/opc-foundation/history/

11 Unified Architecture [Internet]. OPC Foundation. [cited 2023 May 28]. Available at: https://opcfoundation.org/about/opc-technologies/opc-ua/

12 C++ Based OPC UA Client/Server/PubSub SDK: Address Space Concepts [Internet]. [cited 2023 Oct 27]. Available at: https://documentation.unified-automation.com/uasdkcpp/1.7.3/html/L2UaAddressSpaceConcepts.html#L3UaAdrSpaceConceptNodeModel_nodeclasses

13 C++ Based OPC UA Client/Server/PubSub SDK: OPC Unified Architecture Overview [Internet]. [cited 2023 July 17]. Available at:

https://documentation.unified-automation.com/uasdkcpp/1.7.3/html/L2OpcUaFundamentalsOverview.html

14 Key OPC UA Security Concepts [Internet]. [cited 2023 August 8]. Available at: https://www.ptc.com/en/blogs/iiot/opc-ua-security

15 OPC UA Redefines Automation Architectures [Internet]. Automation.com. [cited 2023 October 27]. Available at: https://www.automation.com/en-us/articles/2011-1/opc-ua-redefines-automation-architectures