Cape Peninsula
University of Technology

SMART METER COMMUNICATION AND CONTROL IN AN IEC 61850 BASED ENVIRONMENT

by

RYAN KRUGER

Thesis submitted in partial fulfilment of the requirements for the degree

Master of Engineering: Electrical Engineering: Smart Grid

in the Faculty of Engineering and Build Environment

at the Cape Peninsula University of Technology

Supervisor: Dr. Carl Kriger

Bellville Campus, Cape Town

Date Submited: 13th December 2024

DECLARATION

I, type your full first names and surname here, declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

13-02-2025

Signed **Date**

ACKNOWLEDGEMENTS

- I would firstly like to thank my beautiful and courageous partner Corli for always motivating me to continue and give my best.
- I would also like to thank my parents and family for providing me with the means to attempt this feat, without them none of this would be possible.
- I would like to thank Dr. Carl for his endless patience, guidance and inspiration.

DEDICATION

This thesis is dedicated to my late mother, Annelize Botha Van Tonder, without whom this achievement would not have been possible.

ABSTRACT

The rapid evolution of energy systems has driven the need for advanced communication and management technologies within the power grid. The technological developments and advancements have resulted in a complex power system requiring protection, automation, monitoring and control strategies and methodologies to ensure its efficient, safe and continued operation. The various devices required for the protection, automation, monitoring and control are generally from different manufacturers and are all required to function together in this system known as the Smart Grid (SG). Challenges of interoperability between these devices have necessitated the development of communication standards such as the IEC 61850 standard that have interoperability and future proofing as its primary drivers.

This thesis focuses on integrating the IEC 61850 standard into smart metering systems to address the critical challenges of interoperability, scalability, and efficient data exchange in distributed energy environments. The research explores the potential of the Manufacturing Message Specification (MMS) protocol to standardize communication frameworks and enhance the reliability of SG operations.

The study employs a modular design approach, incorporating open-source tools, Raspberry Pi hardware, and Dockerized environments to develop a scalable and adaptable smart metering solution. The methodology spans planning, implementation, and testing phases, with rigorous evaluations conducted to validate the system's performance under various conditions. Key findings demonstrate the system's ability to achieve seamless communication between diverse devices while maintaining high levels of efficiency and reliability.

The results contribute to the field by providing a practical framework for applying the IEC 61850 standard outside of traditional substation environments. It highlights the advantages of standardized protocols in reducing operational complexities by eliminating communication interoperability issues and supporting sustainable energy practices. The study also identifies limitations, such as scalability in larger deployments and security considerations, and proposes directions for future research to further optimize the system.

The results also contribute to advancing the SM design by combining existing technologies with the IEC 61850 standard, thus providing a comprehensive framework

for modern smart metering systems that extend beyond traditional substation applications.

The results additionally contribute to the integration of IoT and Web Services within an IEC 61850 environment and demonstrating new pathways for integrating diverse technologies into the SG.

Finally, the thesis findings contribute to device standardization and interoperability thus reducing complexity and enabling seamless device integration in distributed energy networks.

In conclusion, this thesis underscores the transformative potential of integrating standardized communication protocols into SG technologies, paving the way for more resilient and efficient energy systems.

Table of Contents

Table of Figures

Table of Tables:

Glossary of terms

ACSI - Advanced Communication System Interface

API - Application Programming Interface

ASN.1 - Abstract Syntax Notation One

CORBA - Common Object Request Broker Architecture

COaP – Constrained Application Protocol

CPU – Central Processing Unit

DCOM - Distributed Component Object Model

GOOSE - Generic Object-Oriented Substation Event

GPIO – General purpose input output

GPU – Graphical Processing Unit

GUID – Global Unique Identifier

HEMS - Home Energy Management System

HTTP - Hypertext Transfer Protocol

ICT - Information and Communication Technology

IEC – Internation Electrotechnical Commission

IoT - Internet of Things

MMS - Manufacturing Message Specification

MCU – Micro Controller Unit

MQTT - Message Queuing Telemetry Transport

NIC – Network Interface Card

OOD - Object-Oriented Design

OOP - Object-Oriented Programming

RFID - Radio-Frequency Identification

RDBMS – Relational Database Management System

SCL - Substation Configuration Language

SCADA - Supervisory Control and Data Acquisition

SG – Smart Grid

SM - Social Media

SOAP - Simple Object Access Protocol

SPI – Serial Peripheral Interface

SSID – Services Set Identifier

TCP/IP - Transmission Control Protocol/Internet Protocol

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

VMD - Virtual Manufacturing Device

CHAPTER 1
INTRODUCTION

1.1 Background and Context

The rapid transformation of energy systems worldwide has started the development of smart technologies capable of meeting the growing demands for efficiency, reliability, and sustainability. The concept of the Smart Grid (SG), which incorporates advanced communication and control mechanisms, has emerged as a cornerstone of modern power systems. At the heart of this transformation lies the integration of smart metering systems, which not only facilitate real-time energy monitoring but also enable bidirectional communication between consumers and utilities to maintain the balance between energy supply and demand.

In this context, the IEC 61850 standard has garnered significant attention due to its standardized communication in electrical systems built on a standardized data model and standardized configuration language. Originally designed for substation automation, the standard has been adapted for broader applications, including the incorporation of SMs in power grids. By leveraging the Manufacturing Message Specification (MMS) protocol, the IEC 61850 standard ensures interoperability among devices from diverse manufacturers, thereby addressing one of the most critical challenges in modern energy systems. The IEC 61850 standard however has not yet been widely integrated into the realm of cloud computing nor the Internet of Things (IoT) world, although there are limited research projects in this arena (Iglesias-Urkia, et al., 2019) .

This research projects attempts to bridge the gap between the IEC 61850 and recent innovations in the technology industry.

1.2 Problem Statement

Despite the advancements in smart metering technologies, the lack of standardized communication frameworks poses significant barriers to their widespread adoption.

Existing solutions often rely on proprietary protocols, which limit interoperability and hinder the seamless integration of devices in a distributed power grid environment. This fragmentation not only increases operational complexity but also undermines the potential benefits of smart metering systems in terms of efficiency and scalability.

This research seeks to address this gap by implementing a smart metering system that integrates the IEC 61850 standard. Specifically, it focuses on leveraging the MMS protocol to facilitate standardized communication and data exchange in a SG setting.

1.3 Research Aim and Objectives

The primary aim of this research is to design, implement, and test a smart metering system that adheres to the IEC 61850 standard, with a focus on achieving interoperability and scalability using the Ethernet medium.

Theoretical Objectives:

- To conduct a literature in MMS protocol and the IEC 61850 standard uses in the SG and previous integrations with web services.
- To conduct a literature review on the design and applications of residential SMs to determine the technologies used to create a smart meter system.
- To analyse the applicability of the IEC 61850 MMS protocol in smart metering systems and identify its advantages in enhancing communication within distributed energy environments.
- To explore the object-oriented capabilities of the IEC 61850 standard and how they contribute to achieving standardization in SG communication.
- To investigate the implications of adopting international standards like IEC 61850 in achieving scalability and reliability in smart metering systems making use of cloud computing.

Practical Objectives:

- To implement a smart meter network (that represent how a residential smart meter system will function in a centralized manor) that integrates with an IEC61850 standard based system using the MMS protocol as interface.
- To utilize Open-Source Technologies for Cost-Effective Deployment.
- To deploy Modular and Scalable System Architecture using containerization.

1.4 Research questions

- Can SMs situated in a residential environment integrate with an IEC 61850 standard based system?
- Can SMs make use of open-source components and still provide the benefits of a proprietary SM?
- Can cloud computing make integration between different standardized possible or easier?

- Can the IEC 61850 standard be applicable outside the station and how is it done?

1.5 Hypothesis

- SMs situated in residential environments can be effectively integrated with IEC 61850 standard-based systems to enable standardized communication and real-time data exchange, overcoming the limitations of proprietary protocols.
- Open-source hardware and software components can replicate the functionality, scalability, and reliability of proprietary SMs, providing a cost-effective alternative without compromising performance.
- Cloud computing can facilitate the integration of diverse communication standards, such as IEC 61850, by enabling centralized data storage, processing, and seamless interoperability between distributed devices.
- The IEC 61850 standard can be adapted for use outside substation environments, specifically in residential SG systems, through the use of modular design approaches, containerized environments, and advanced networking techniques.

1.6 Significance of the Study

The findings of this research are expected to contribute significantly to the field of SG technologies. By demonstrating the feasibility of integrating the IEC 61850 standard into smart metering systems, this study provides a pathway for achieving standardized communication in distributed energy systems. The proposed solution also holds potential for enhancing grid reliability, reducing operational costs, and promoting the adoption of renewable energy sources through better energy management practices.

1.7 Methodology Overview

This research adopts a methodology that is used in software development life cycles (SDLC). The methodology is based around building and integrating these components. Each component will mainly be software components, hence the adoption of this methodology. The SDLC consists of five sections:

- Requirements gathering - This section is based around requirements gathering.
- Planning and Design - Designing the new system and each of the components.
- Implementation - The execution of the deliverables from the “Planning and Design Phase”

- Testing - The planning phase outlines acceptance criteria, which is required for the system to be label as successful. This phase tests the system to determine if the acceptance criteria is met.
- Reflection – The reflection phase is mainly reflecting on the project concluding the downloads.

The discussed methodology outlined affect the structure of this research report. The structure of this report can be seen in the following section

1.8 Structure of the Thesis

This thesis is organized into several chapters, each addressing a specific aspect of the research:

- Chapter 1: Introduction – Provides the background, problem statement, objectives, and significance of the study.
- Chapter 2: Literature Review – Explores existing research on smart metering systems and the IEC 61850 standard.
- Chapter 3: Research Methodology and planning phase – Details the planning and design of the proposed system.
- Chapter 4: Implementation and integration of the SM – Describes the technical aspects of system development and integration.
- Chapter 5: Results and Testing – Documents the testing process and analyses the system's performance.
- Chapter 6: Conclusion – Summarizes the findings, discusses implications, and suggests future research directions.

1.9 Delimitations

While this research provides a significant contribution to the integration of the IEC 61850 standard into smart metering systems, several delimitations must be acknowledged to contextualize its findings and identify areas for improvement:

- Controlled testing environment

  The testing and validation environment is to be done within a software development environment and does not include testing using real world load data on the entire system. A larger approach might be required to test and handle the impact of a real-world load on the system.

- Lack of Substation Integration

While this system tests the integration of IEC 61850 standard-based approved protocols like MMS, the integration of such information into an actual SCADA system will not been tested.

- Security

Some of the software packages used in this research have not been updated for some time. Hence the use of this software might present security issues if used in a real-world setting.

- Lack of hardware range

Only a few IoT devices are to be used in this research which might create a lack of range. The Raspberry Pi will be used in this research project, which is quite a powerful device. Lower-level microcontroller units will not be used in this work.

1.10 Conclusion

In summary, this thesis addresses the critical need for interoperability and standardization in SG communication by integrating the IEC 61850 standard into smart metering systems. The study's objectives focus on leveraging the MMS protocol to enhance scalability, reliability, and efficiency in distributed energy systems. By outlining the research problem, objectives, and methodology, this chapter establishes a solid foundation for the subsequent exploration and implementation detailed in the following chapters.

The next chapter delves into a comprehensive literature review, examining existing smart metering technologies, communication frameworks, and the role of the IEC 61850 standard. It identifies gaps in current research and lays the theoretical groundwork for the proposed methodology.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

This chapter presents a comprehensive overview of the existing knowledge on Smart Meters (SM), communication infrastructure of the power grid, the IEC 61850 standard, and Application Programming Interfaces (APIs) in power grids through a literature review. The IEC 61850 standard is an international standard for communication in electric power systems and is designed to provide a common communication protocol and data model for substation automation and Smart Grid (SG) communication. The goal of the IEC 61850 standard is to ensure interoperability between equipment from different manufacturers in substations.

The aim of this research work as stated in Chapter 1 is to incorporate the IEC 61850 standard to control and communicate with the load using SMs, outside of the substation. To enable us to advocate for the IEC 61850 standard; existing standards must first be investigated. Existing standards that work to complete the same goal as the IEC 61850 standard, are investigated in this section.

This section reviews existing published literature. This section is split into three subsections namely the IEC 61850 standard with the focus on the MMS (Manufacturing Message Specification), web service architecture integration with an IEC 61850 standard environment and SM design. The goal of this section is to showcase where advancements in technology can be made, that have not been researched.

This chapter consists of three sections of those sections each having its own sub sections. Section 2.2 reviews the literature on the IEC 61850 standard which is used in this research work, focussing on the MMS. Section 2.3 introduces and reviews literature that cover topics regarding web services in the SG. Section 2.4 focusses on the client that this research work proposes, which is the SM.

2.2 Literature review of the MMS and the IEC 61850 standard

2.2.1 Introduction

The IEC 61850 standard is an international standard that provides a comprehensive framework for the design, configuration, and communication of digital substations in the power industry. It presents a unique exchange of information between different components within the substation, including the way that it standardizes protection and control system, electrical equipment, and communication network functions. The result is a more interoperable and efficient substation system that reduces the risk of errors

and improves overall performance. The IEC 61850 standard provides a common language for all components in a substation, which leads to increased system reliability, improved maintenance, and reduced costs. Additionally, its flexible structure allows for easy integration with other systems, making it a future-proof solution (based on Ethernet communication) for the evolving demands of the power system industry.

The IEC 61850 standard makes use of a standardized data object model, which uses an object-oriented approach. This is different to Modbus and DNP3 communication which uses indexes to acquire/send data. Thus, creating a large set of data being sent around for a singular value. Using the OOD (Object Oriented design) approach, the IEC 61850 standard breaks down systems in layers ranging in complexity. The Service device – which is the highest layer of a device hierarchy – can be broken down into the smallest building element called logical nodes, which can have different functions and data depending on the type of logical nodes. This approach makes it very easy to integrate with software development life cycles, because most programming language incorporates OOP (Object Oriented Programming) into their core (Haung, 2018).

The next section details the purpose of the IEC 61850 standard and reviews the model and services that the standard provides.

2.2.2 Advantages of IEC61850

IEC 61850 has several key advantages over competing standards:

2.2.2.1 Interoperability and Standardization

IEC 61850 was designed to enable interoperability between devices from different manufacturers. Unlike Modbus and DNP3, which require customized mappings and adaptations, IEC 61850's object-oriented data modeling ensures seamless communication and integration across various devices and systems (Ağin, et al., 2024).

2.2.2.2 High-Speed Communication

IEC 61850 employs advanced communication protocols such as GOOSE (Generic Object-Oriented Substation Event) and MMS (Manufacturing Message Specification), which provide real-time data exchange with minimal latency. This is particularly important for time-sensitive substation automation functions that require fast response times (Apostolov, 2017).

2.2.2.3 Scalability and Future-Proofing

Unlike traditional protocols that were primarily designed for specific tasks, IEC 61850 was built with a flexible architecture that can accommodate future expansions, including the integration of IoT, cloud computing, and distributed energy resources

(DERs). It is also highly adaptable to modern networking technologies like Ethernet-based communication (Park, et al., 2012).

2.2.2.4 Object-Oriented Data Modeling

IEC 61850 uses an advanced hierarchical data modeling approach, representing devices as logical nodes with standardized attributes. This object-oriented design allows for efficient data organization and retrieval, reducing the complexity of managing large-scale power systems.

2.2.2.5 Security and Reliability

Security is a crucial aspect of IEC 61850, as it supports authentication and encryption mechanisms under IEC 62351. This contrasts with Modbus and DNP3, which lack built-in security features, making them more vulnerable to cyber threats (Hussain, et al., 2023).

2.2.3 Competing Standards

Despite its advantages, IEC 61850 faces competition from several established communication standards in the power industry.

2.2.3.1 Modbus

Modbus is a widely used communication protocol known for its simplicity and ease of implementation. However, it lacks the high-speed performance, scalability, and security features of IEC 61850. Modbus relies on a polling mechanism, which can lead to inefficient bandwidth utilization and slower response times (Ağin, et al., 2024).

2.2.3.2 DNP3 (Distributed Network Protocol)

DNP3 is a robust protocol commonly used for SCADA (Supervisory Control and Data Acquisition) systems. It offers better security and reliability than Modbus but still falls short of IEC 61850 in terms of flexibility and real-time data exchange capabilities. Unlike IEC 61850's event-driven approach, DNP3 primarily relies on polling mechanisms, making it less efficient for high-speed automation (Wang, et al., 2008).

2.2.3.3 Smart Meter Language (SML)

Smart Meter Language (SML) is a communication protocol primarily used in smart metering applications. SML facilitates efficient data transmission between smart meters and energy management systems. While it is lightweight and optimized for metering applications, it lacks the scalability and broader interoperability provided by IEC 61850. Unlike IEC 61850, which is designed for extensive substation automation and grid-wide communication, SML is limited to specific smart metering use cases. Furthermore, SML does not incorporate the advanced object-oriented data modeling

and high-speed event-driven communication that IEC 61850 offers (Burunkaya & Pars, 2017).

2.2.3.4 Proprietary Protocols

Many vendors develop proprietary communication protocols tailored to their products. While these protocols may offer optimized performance for specific applications, they create vendor lock-in issues and limit interoperability with third-party devices. IEC 61850 eliminates this problem by providing an open, standardized framework (Iglesias-Urkia, et al., 2018).

2.2.4 Abstract Communications Service Interface (ACSI)

The IEC 61850 standard introduces the ACSI model and services. These services are created to dictate the *how*, *what,* and *where* with regards to the capability of an IEC-61850 compliant server. The IEC 61850-8-1 part of the standard documents the mapping to MMS (IEC, 2004).

Table 2.1: List of ACSI classes (IEC, 2003)

SERVER model (Clause 6) GetServerDirectory **ASSOCIATION model (Clause 7)** Associate Abort Release **LOGICAL-DEVICE model (Clause 8)** GetLogicalDeviceDirectory **LOGICAL-NODE model (Clause 9)** GetLogicalNodeDirectory GetAllDataValues **DATA model (Clause 10)** GetDataValues SetDataValues GetDataDirectory GetDataDefinition **DATA-SET model (Clause 11)** GetDataSetValues SetDataSetValues CreateDataSet DeleteDataSet GetDataSetDirectory **Substitution model (Clause 12)** SetDataValues GetDataValues **SETTING-GROUP-CONTROL-BLOCK model (Clause 13)** SelectActiveSG SelectEditSG SetSGValues ConfirmEditSGValues GetSGValues GetSGCBValues **REPORT-CONTROL-BLOCK and LOG-CONTROL-BLOCK model (Clause 14)** **BUFFERED-REPORT-CONTROL-BLOCK:** Report GetBRCBValues SetBRCBValues **UNBUFFERED-REPORT-CONTROL-BLOCK:** Report GetURCBValues SetURCBValues	**LOG-CONTROL-BLOCK model:** GetLCBValues SetLCBValues QueryLogByTime QueryLogAfter GetLogStatusValues **Generic substation event model – GSE (Clause 15)** **GOOSE** SendGOOSEMessage GetGoReference GetGOOSEElementNumber GetGoCBValues SetGoCBValues **GSSE** SendGSSEMessage GetGsReference GetGSSEDataOffset GetGsCBValues SetGsCBValues **Transmission of sampled values model (Clause 16)** **MULTICAST-SAMPLE-VALUE-CONTROL-BLOCK:** SendMSVMessage GetMSVCBValues SetMSVCBValues **UNICAST-SAMPLE-VALUE-CONTROL-BLOCK:** SendUSVMessage GetUSVCBValues SetUSVCBValues **Control model (Clause 17)** Select SelectWithValue Cancel Operate CommandTermination TimeActivatedOperate **Time and time synchronization (Clause 18)** TimeSynchronization **FILE transfer model (Clause 20)** GetFile SetFile DeleteFile GetFileAttributeValues

Table 2.1 shows a list of services outlined to interact with the server model. These services look very similar to methods used in Object Oriented Programming (OOP).

The second class from Table 2.1 being exposed is the association model. The association model describes how to integrate communication architecture like the publisher-subscriber architecture or client-server architectures. Access control concepts are also discussed to ensure the security of a server.

The ACSI outlines how to use the object references to identify and find data in a standardized way. It is like giving every piece of information its own special address to enable external devices to communicate effectively and share the right information with each other. Figure 2.1 shows an example of an object reference. To keep these references unique the interface also outlines requisite rules (Ağin, et al., 2024).

Figure 2.1: Example of the object reference (IEC, 2003)

Outlined in Table 2.2 is the ACSI (Abstract Communication Service Interface) services associated with the data models the IEC 61850 standard provides. The services outlined below are the main services that interact with the server models and the hierarchy of the server. The other services are also important, but these allow an external party to interact with a server directly via a network connection. The table also outlines the requisite parameters required to get the required response.

Table 2.2: List of Service outlined by ACSI to interact with the server model (IEC, 2004).

Model	Services	Description of service	Request parameter	Response parameter
Server	GetServerDirectory	Retrieves Logical devices and files	ObjectClass	LDRef [1……n] Files
Logical device	GetLogicalDeviceDirectory	Retrieves a specific logical device, with all the logical nodes associated to that logical device.	LDRef	LDName LDRef LNRef [3…n]
Logical Node	GetLogicalNodeDirectory	Retrieves a list of all objects references visible by requesting client.	LNRef	LNInstance
	GetAllDataValues	Retrieves all data Attributes	LNRef Functional constraint	LNRef DataAttibute[1..n] DataValue[1…n]

Model	Services	Description of service	Request parameter	Response parameter
Data Model	SetDataValues	Write value to data model in logical node.	Data Reference Functional constraint DA Component name Values	Ok.
	GetDataValues	Reads values of a data model contained in logical node.	Data Reference Functional constraint DA Component name	DataAttribute [1…n]
	GetDataDirectory	Retrieves object references of all DA's.	DataReference	Data Attribute [1…n]
	GetDataDefinition	Retrieves definitions of all DA in Data model	DataReference	Data Attribute [1…n]
Data Set Model	GetDataSetValues	Retrieves a DataSet	DataSetReference	DataSetReference DataAttributeValue [1...n]
	SetDataSetValues	Sets value of a DataSet	DataSetReference DataAttributeValue [1...n]	Status Code: Ok
	CreateDataSet	Creates a custom Data Set	DataSetReference DSMemberRef [1...n]	
	DeleteDataSet	Deletes a data set	DataSetReference	Status Code: Ok
	GetDataSetDirectory	Retrieves Dataset references	DataSetReference	DSMemberRef [1...n]

2.2.5 MMS ACSI extension

Table 2.3: List of services outlined by MMS protocol (Park, et al., 2012)

TAG	Service
0	status
1	getNameList
2	identify
3	rename
4	read
5	write
6	getVariableAccessAttributes
7	defineNamedVariable
8	defineScatteredAccess
9	getScatteredAccessAttribute
10	deleteVatiableAccess
11	defineNamedVariableList
12	getNamedVariableListAttributes
75	fileRename
76	fileDelete
77	fileDirectory

Table 2.3 outlines a list of services the MMS protocol outlines. The IEC 61850 standard has its own set of services it outlines for various object classes. The above services allow MMS to transmit real-time data between client and server (Park, et al., 2012).

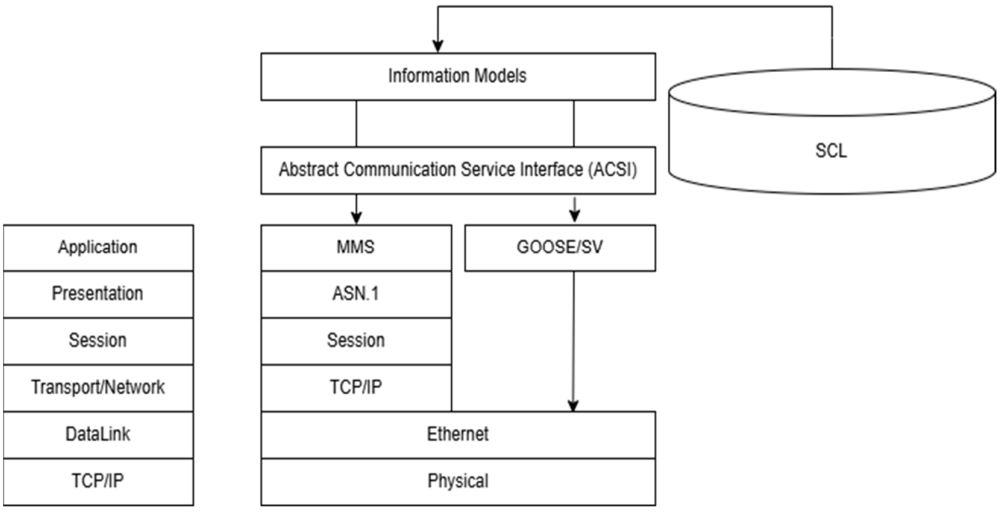

Figure 2.2: Communication Structure of IEC 61850 (Park, et al., 2012)

Figure 2.2 shows the MMS server uses the entire stack of the OSI layer, whereas GOOSE (Generic Object-Oriented Substation Event) and SV (sampled values) uses only two layers namely the Physical and Datalink layers. The ACSI layer requires a SCL (Substation Configuration Language) file to provide models that are used to communicate with. The MMS mapping provides real time read/write capabilities to other networked devices and provides a client/server communication architecture (Park, et al., 2012).

Table 2.4 shows the mapping of MMS classes to the IEC 61850 classes along with the services that the MMS provides for each of those classes.

Table 2.4: Mapping of MMS object to that of IEC 61850 (IEC, 2004)

MMS OBJECT	IEC 61850 OBJECT	MMS SERVICES IN USE
Application Process VMD	Server	Initiate Conclude Abort Reject Cancel Identify[1]
Named Variable Objects	Logical Nodes and Data	Read Write InformationReport GetVariableAccessAttribute GetNameList
Named Variable List Objects	Data Sets	GetNamedVariableListAttributes GetNameList DefineNamedVariableList DeleteNamedVariableList GetNameList Read Write InformationReport
Journal Objects	Logs	ReadJournal InitializeJournal GetNameList
Domain Objects	Logical Devices	GetNameList GetDomainAttributes StoreDomainContents
Files	Files	FileOpen FileRead ObtainFile FileClose FileDirectory FileDelete

Table 2.4 shows that the MMS server has services: Initiate, abort and conclude. Those services map one-to-one to that of the IEC 61850 standard which has a data model. The service in question is the service responsible for creating connections with clients.

Park, et al., (2012) created an object-oriented approach to implementing IEC 61850 MMS mapping on a PC with a Windows operating system. Making use of C++, a list of reusable classes is implemented to simulate the entire standard. The researchers created a layered approach to simulating the classes.

Table 2.5: List of classes implemented to simulate MMS-to-IEC 61850 mapping (Park, et al., 2012).

Name	Description
MMSClientReq	Functions to request data or operational information to server
MMSServerRsp	Functions to response the requested behavior to client
MMSCOMMREQServ	Functions to make messages for requesting to server
MMSCOMMRSPServ	Functions to make messages for responsing to client
MMSDecoder	Functions to decode MMS messages
MMSOpcode	Functions to get demanded information by result of MMSDecoder layer
MMSMpl	Functions to make MMS messages for requesting/responsing
REQControl	Functions to manage request messages and check whether the response messages are recieved or not
ASN1	Functions to encode or decode response/request messages to ASN.1 format
COMMServ	Functions to set or free the connection and take responsibility for receiving and sending packets
ACSEServ	Functions to provide environment of client-server type program
ISOPP, ISOSP, ISOTP	Functions to negotiate the transfer syntaxes, manage session and check error
GeneralSocket	Functions to manage packets for actual communication

Table 2.5 shows a list of classes, and their description implemented by Park, et al., (2012). MMSClientReq, MMSSErverRsp, MMSCOMMREQServ, MMSCOMRESServ, MMSDecoder, MMSOpCode, MMSMpl, REQControl are all created to function as the MMS application layers.

The IEC 61850 standard has another communication type it introduces other than the MMS. The Generic Object-Oriented Substation Event (GOOSE) is a peer-to-peer protocol within the IEC standard that is completely different to that of MMS but provides great benefits for substation communication.

2.2.6 Generic Object-Oriented Substation Event (GOOSE)

As mentioned above the GOOSE protocol introduces a new different way of communicating. The 2-layer approach to transmitting data enables devices within substations to exchange data in a time-critical manner. The two or three layers of the of the OSI model are used instead of all seven layers and makes it ideal for protection applications as the speed of transmission is increased. Combined with the object-oriented approach that the GOOSE protocols use, it allows IEC 61850 data models to easily access information within the substation without the trouble of first establishing a connection with the server device. This is improved upon by introducing a publisher-subscriber architecture, allowing equipment to be subscribe to different devices and their associated functions within the substation (Kriger, et al., 2013).

The introduction of R-GOOSE (Routable GOOSE) and R-SV (Routable Sampled values) added to the GOOSE protocol, allow for inter-substation communication. The network and transport layers allow GOOSE messages to be routed between different substations, creating routable GOOSE messages (Hussain, et al., 2023).

The protocol is documented to be used within the substation and in later revisions the generation. This research work focusses on using IEC 61850 on part of the power grid that does not allow the connection to be unsecure. Using a 2-Layered approach to communication removes the layers associated to securing a connection and ensuring the connection (Apostolov, 2017). Making use of the seven layers of the OSI does have performance draw backs, but a more trusted connection is introduced.

Although R-GOOSE allows GOOSE messages outside the substation, this research work focusses on non-substation related communication. Making use of either R-GOOSE or GOOSE which creates various security issues for a residential related area using GOOSE/R-GOOSE (Hussain, et al., 2019).

2.2.7 Literature review of the MMS protocol

2.2.7.1 Introduction

The MMS is an international standard for exchanging real-time data between devices and computer applications in an industrial environment. It was created to play the role of interoperable communication protocol between manufacturing equipment. MMS is a TCP/IP (Transmission Control Protocol/Internet Protocol) based protocol that is used

to communicate with different components in a substation on station and bay levels. The MMS protocol can also be used with a UDP (Unified datagram protocol) to allow faster communication (Ruland, et al., 2016). The standard can also be extended to work as a gateway interface to a substation (Hou, et al., 2013).

This literature review focusses on the advancement and integration of the MMS standard not only within an IEC 61850 environment but also in a web service environment. Two major topics are investigated in this literature review of the MMS; mutation and integration are the main topics related to the use of MMS.

2.2.7.2 Purpose of the MMS

The MMS allows for the retrieval, manipulation, and dissemination of real-time and historical data, as well as the control and configuration of devices, all while ensuring data integrity, reliability, and security.

The MMS was created to allow interoperable communication between manufacturing equipment. Making use of an object-oriented approach the protocol was adopted by the IEC 61850 standard. In an IEC 61850 environment the MMS protocol is used to communicate and control and actions within a substation environment. The protocol's use of TCP/IP allows it to function in more than just a single substation. As a result, MMS empowers power utilities and industries to optimize their operations, enhance maintenance strategies, and facilitate informed decision-making through streamlined communication and data accessibility. It allows the Supervisory Control and Data Acquisition (SCADA) system in a control centre to connect to individual substation hierarchies to get reporting information.

2.2.7.3 Virtual Machine Device model

The MMS protocol was originally published in 1988. The MMS standard consists of two parts. Part 1 introduces the Virtual Manufacturing Device (VMD) model, and the services associated with that model. Part 2 introduces the communication specification of messages and the interaction of MMS with the other layers of the OSI model. The standard makes use of ASN.1 (Abstract Syntax notation one) to format the MMS messages (SISCO, 1995).

The MMS's key feature is the virtualizing a device, very similar to the IEC 61850 server model. The MMS was created to represent industrial equipment as VMD's (Virtual Manufacturing Device). The MMS outlines services that clients use to interact with MMS devices/MMS servers. The MMS outlines the behaviour a server has upon request (SISCO, 1995).

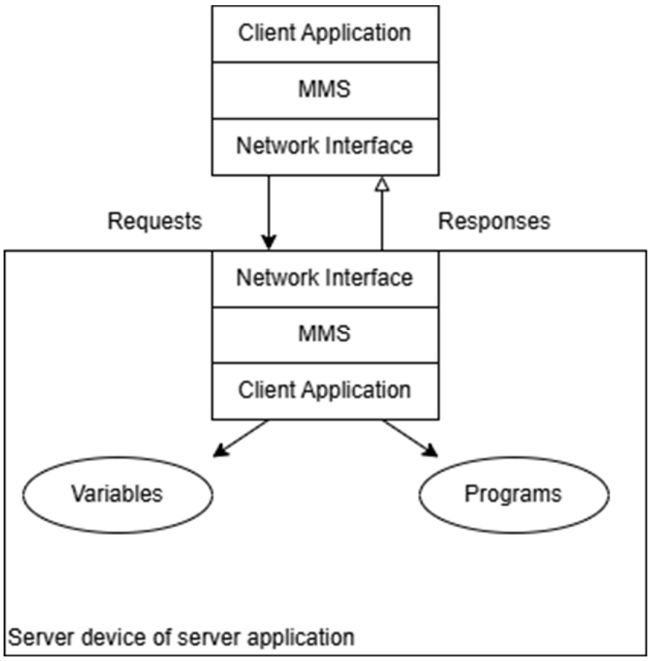

Figure 2.3: Communication with MMS protocol (SISCO, 1995)

From Figure 2.3, both the client and the VMD (server) have a layer in their software assigned to the MMS. Each one of the devices is responsible for the formatting of the data either going or coming from the respective devices via their individual application layer. The MMS provides the application layer to these applications and any application that would need to use this information would also require an MMS application layer to read/write the data.

The IEC 61850 standard outlines the mapping procedures between ACSI and MMS in part 8.1 of the standard. The IEC 61850 standard makes use of its object-oriented approach, instead of the linear approach like non-object-oriented standards like DNP3. The MMS makes use of a very similar model. The IEC 61850 standard outlines a Server, Logical device, logical node, data set and data attribute. The MMS standard outlines VMD, domain, named variables, named variable list, journal and file management as interactable models that represent a device (Wang, et al., 2008)

2.2.7.4 MMS system integration

There are various ways of integrating the MMS interface into a power grid related system. Providing an already existing system with the MMS interface and connecting it to the architecture can allow a non-IEC 61850 compliant system to become IEC 61850 compliant.

MMS allows the transfer of IEC 61850 data over the Internet. Making use of the full OSI and TCP/IP the MMS protocol can be used between geographically distanced entities. The client-server communication architecture allows different clients to connect to the

MMS server using a gateway. Hou, et al., (2013) created a MMS application gateway, by hosting a server that makes use of a substation configuration file, allowing the retrieval of real-time data using a MMS client server.

2.2.7.5 Breakdown of literature review

Below is a figure of papers that were reviewed to determine the implementation and mutation of MMS between 1995 and 2023. Such a large date range has been covered to determine how researchers have either changed to the MMS protocol to become better and how they integrate already existing systems with the MMS standard to determine the conversion from proprietary to standardized systems.

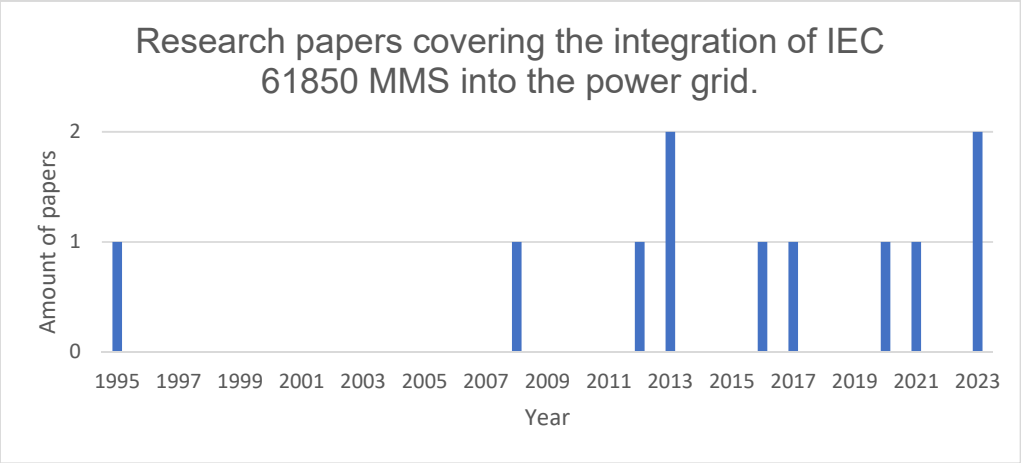

Figure 2.4: Bar graph showing the spread of research papers for keywords.

Figure 2.4 shows the spread of research papers regarding keywords: IEC 61850 MMS Integration, IEC 61850 MMS webservice Integration and MMS mutation. From the table 2.2 it can be seen there has not been a prominent researcher covering research with relation to the searched keywords.

The papers used cover a very specific set papers that all cover IEC 61850, MMS and the power grid. A paper from 1995 is included in the compilation of results. This paper mainly covers MMS in the power grid. This is the only exception that was included in this portion of the report. The IEC 61850 standard was published in 2003 and therefore we do not see any contributing research in the area of interest for the next few years following its publication.

2.2.7.6 Comparative analysis of IEC 61850 standard literature

2.2.7.6.1 Comparative analysis of system integration of the MMS standard.

The below table contains a list of papers where the researchers implemented the MMS integration within a system.

Table 2.6: Research papers covering the implementation of the MMS standard

Paper	Title	Description of project	Hardware Requirements	Software Requirements
(SISCO, 1995)	Overview and introduction to the Mixed Manufacturing Specification	This report documents the use, history, and integration of MMS.	Not specified	Not specified
(Wang, et al., 2008)	Research on Distributed Transmission of Power Telecontrol Information Base on ACSI/MMS	A system is designed and created to allow the integration of MMS devices to connect via a WAN (wide area network). The focus being on the control of various devices from a control centre perspective to a substation.	Not specified	Not specified
(Park, et al., 2012)	IEC 61850 Standard Based MMS communication Stack Design using OOP	The researcher creates a C++ library using the factory design pattern. Making use of the object-oriented approach IEC 61850 uses and using rules of OOP a library able to create servers and clients is constructed.	Not specified	Not specified
(Hou, et al., 2013)	Research on IEC 61850 Gateway based on MMS mapping	The researchers created an abstract gateway that can be used to create future gateways to servers using MMS.	Not specified	Not specified
(Pham, 2013)	Integration of IEC 61850 MMS and LTE to support smart metering communication	The author created an MMS gateway using LTE technology to submit metering data to a MDMS over a large geographical area.	Not specified	Not specified

Paper	Title	Description of project	Hardware Requirements	Software Requirements
(Ruland, et al., 2016)	Rejuvenation of the IEC 61850 Protocol stack for MMS	The researcher attempts to substitute protocols used in the OSI layer of the standard to create a better MMS protocol.	Not specified	Not specified
(Ustun & Hussain, 2017)	IEC 62351-4 Security Implementations for IEC 61850 MMS Messages	To introduce IEC62351-4 security measure to MMS. Making use of TLC an X.509 certificate was developed for both 61850 client and server.	Not specified	Not specified
(Hau-feng, et al., 2020)	Application research on the substitution specification of MMS	Create a new version of the MMS protocol, because the current version is not implementing the ACSI services correctly.	Not specified	Not specified
(Hwang, 2021)	Investigation of Wireless IEC 61850 MMS using Raspberry PI	The researchers create a system using IoT components namely a raspberry Pi, to create a test bed to use MMS wirelessly.	Raspberry Pi	RaspAp
(Oh, et al., 2023)	Design and implementation of IoT(Internet Of Things) Gateway with MQTT and IEC 61850 MMS protocol	Oh et al., 2023, Creates a IEC 61850 MMS test bed using raspberry pi's to create the connection	Raspberry Pi	RaspAp LibIEC61850 BeanItIEC61850

Paper	Title	Description of project	Hardware Requirements	Software Requirements
(Gayo-Abeleira, et al., 2023)	Design and implementation of multiprotocol framework for residential prosumer incorporation in flexibility markets.	A system is created using various IoT protocols including IEC 61850 MMS. The authors make us of a protocol integration platform called FIWARE. FIWARE is used as an intermediary to convert all household power readings from smart plugs, smart TV's, all smart devices using different IoT protocols. The readings is gathered and then converted into a singular database. The system then exposes all information	Various smart devices, i.e. smart plug, smart appliances.	FIWARE Thingsspeak RSCARD

Table 2.6 shows literature that was reviewed and compared to gain an understanding of the application of MMS within and power grid and integration context.

Hwang (2021) and Oh, et al., (2023) Created very similar system adding Message Queuing Telemetry Transport (MQTT) to that which was created by Hwang (2021). Both authors created an IEC 61850 compliant system where they utilize the IEC 61850 MMS protocol to explore the benefits of a wireless connection over the conventional wired connection. The goal of Hwang (2021) is to determine the speed of a connection of a Raspberry Pi (IEC 61850 server) in a wireless network environment. Figure 2.5 below shows the layout of the testbed that is created. All the actors in the diagram below is replaced with Raspberry Pi's, to showcase the creation of a testbed with non-IEC 61850 components

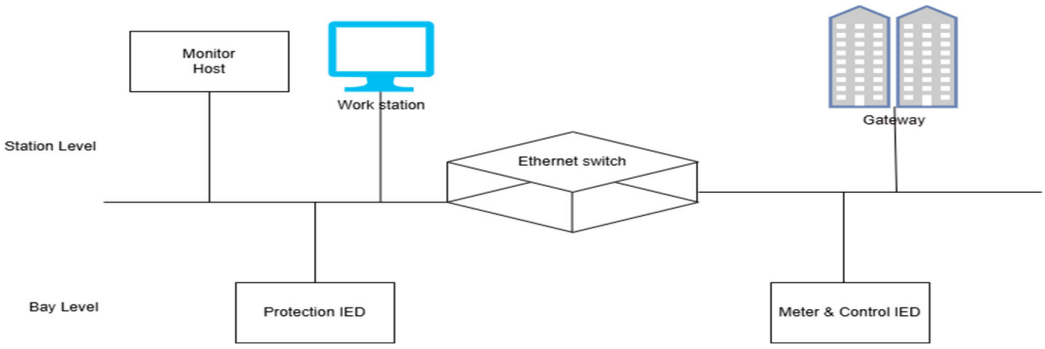

Figure 2.5: Testbed architecture for MMS integration (Haung, 2018)

Oh, et al., (2023) used the below testbed architecture (Figure 2.6) to create an IoT (Internet of Things) gateway to allow the flow MQTT. There are 3 additional roles in the architecture below to that of Hwang (2021). The MQTT Broker, MQTT Subscriber and the MQTT Publisher is introduced into the architecture with the already existing client-server.

The MQTT publisher is the actor which creates topics and generates messages for consumption. The publisher gives all messages and topics to a subscriber.

Operating at the heart of the MQTT architecture, the MQTT Broker serves as the central nexus for message distribution. The broker takes the messages supplied from the publisher and pushes it to the subscribers that are subscribed to that topic.

In the MQTT ecosystem, MQTT Subscribers receives and processes messages of relevance. They subscribe to one or more MQTT topics, thereby receiving messages related to that topic. Subsequently, these Subscribers receive messages that have been published to the topics aligned with their preferences, courtesy of the MQTT Broker (Rusnak, 2022)

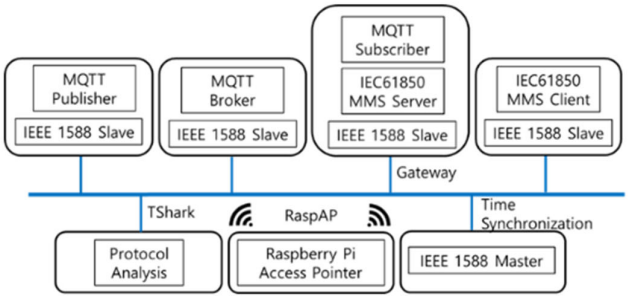

Figure 2.6: Testbed including MQTT actors (Oh, et al., 2023)

(Gayo-Abeleira, et al., 2023), created a framework that integrates IoT devices and IEC 61850 compliant power equipment in a smart connected home scenario. The authors establish various roles and use a central HEMS for communication and ensure compatibility between different devices and standards through the FIWARE platform and IEC 61850 communication. The framework also includes a test platform to validate its functionality.

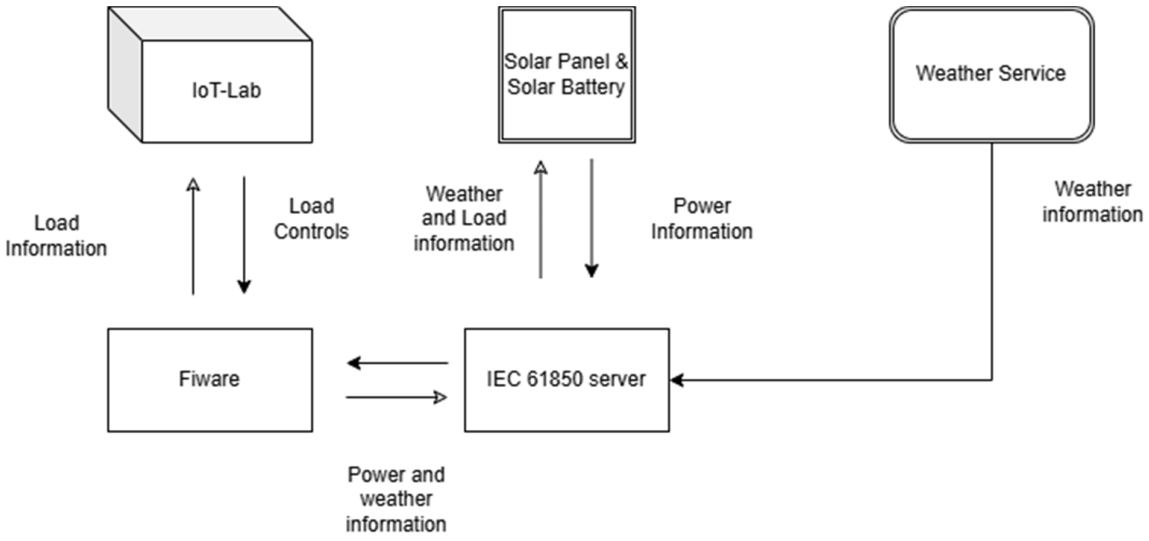

Figure 2.7: Prosumer IoT high level architecture for IEC 61850 data consumption (Gayo-Abeleira, et al., 2023).

Figure 2.7 shows the architecture of all the actors within the authors proposed system. An IEC 61850 server sits between their FIWARE platform. FIWARE is an open-source platform used for an array of standardized elements and resources to craft and implement intelligent applications. It was created for Internet of Things (IoT) to integrate various IoT to interconnect devices of different IoT standards to create and consume real-time data. FIWARE has various protocols it supports; IEC 61850 MMS is amongst one of the supported protocols.

The FIWARE system is responsible for conversion of all the information to the IEC 61850 Server. Figure 2.8 shows the overall connection of the system the author created. The figure also shows the FIWARE system and its IEC 61850 MMS interface connecting to an external server.

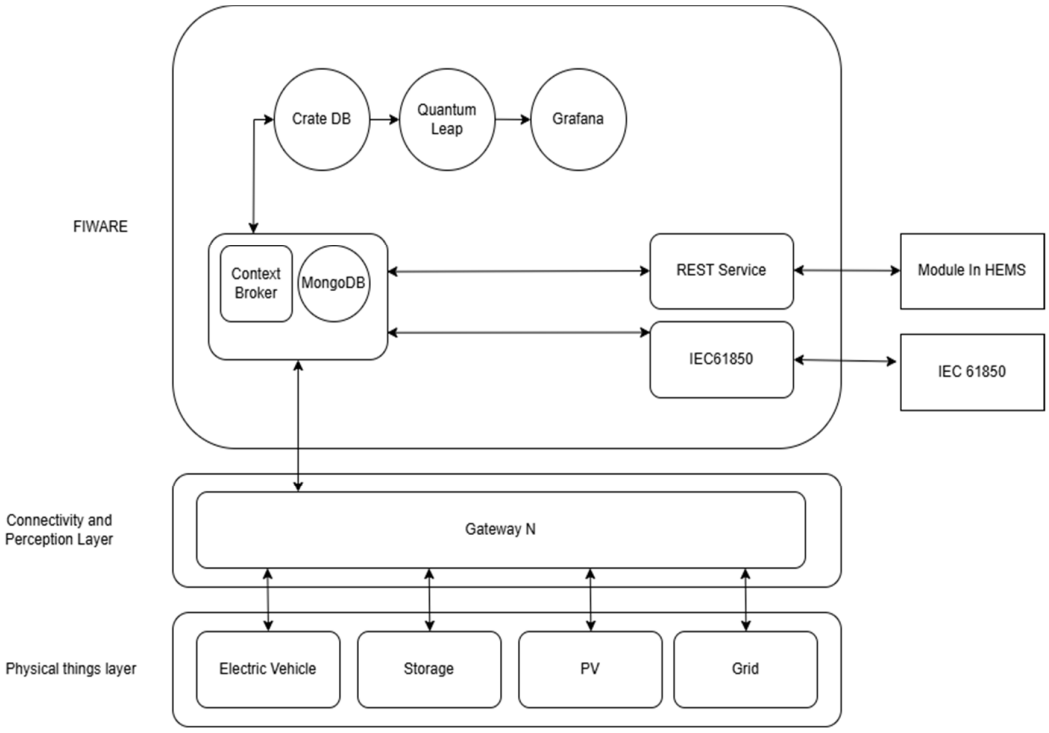

Figure 2.8: Detailed architecture of FIWARE integration (Gayo-Abeleira, et al., 2023).

The FIWARE system makes use of various IoT related protocols. Figure 2.8 shows a perception layer which is used to convert all IoT device meter readings from their various data reading standards to the FIWARE common data standard.

The data is then gathered to and assigned to a context broker that has two responsibilities. The first responsibility is to save the data readings for further consumption. The second data responsibility is to visualize the collected information and to display it to the end user via Grafana (Gayo-Abeleira, et al., 2023).

2.2.7.7 Discussion

From the analysis above the newer research papers focuses more on integrating the MMS systems with IoT standards like: MQTT, REST, lwM2M and LoreWAN. From the papers we can see that it didn't start with a big bang approach but as technology develops the MMS protocol is adapted into different forms.

Wang, et al., (2008), created the first telecontrol system, making use of the MMS. They created a MMS server that can act as a server and client. Making it possible to communicate with a SCADA system without the use of a RTU. Hou, et al., (2013), made use of Wang, et al., (2008)'s implementation of telecontrol and created a framework for a gateway. The framework also used the role of server/client, where the MMS server plays the role of both. Being an intermediary, the MMS server is the client to substations requesting data and a server to devices/components below in the hierarchy.

Authors, Oh, et al., (2023) and Hwang (2022) created a communication infrastructure making use of MMS. Creating a testing infrastructure from Raspberry Pi's to demonstrate the communication between the IEC 61850 standard and various other IoT based standards. They demonstrate, to test the IEC 61850 standard in a safe environment open-source components can be used to emulate the power grid. Oh, et al., (2023), highlighted that the publisher-subscriber architecture can also be set up without the GOOSE protocol, but using the MQTT protocol in conjunction with MMS.

Monitoring each individual device in a house can be cumbersome depending on the manufacturer of that device. If the manufacturer decides to make the device uncontrollable a smart plug is required to make a device more controllable. Gayo-Abeleira, et al., (2023), made use of a IoT communication standard conversion system called FIWARE to monitor each device in a house. Each device having its own IoT interface to communicate with the FIWARE system and create a holistic view of the system being metered. The author only used FIWARE's MMS interface to communicate with IEC 61850 related environments.

Although this project has various devices it monitors within a household this is not feasible because it doesn't cover other power usages like internal resistances. Using the main incoming circuit of a residential house is a better way of monitoring all the power being utilized in a household, because the mains are connected to all the loads that is going into the house via an earth leakage switch.

2.3 Literature of Web Service Architecture with IEC 61850

2.3.1 Introduction

One of the main components of the SG is the use of Information and Communication Technology (ICT), especially the communication aspect. As ICT has progressed over the last 30 years, ways of having computers communicate with each other has also evolved. Instead of connecting via the original peer-to-peer architecture, the client-server architecture has also been widely implemented. Various hybrid communication models have been created based on these technologies. Web services a client-server communication model which is used to host applications or filesystems has made a great contribution to the Internet.

This section discusses the review of literature from authors that have used web services in their papers to create a better SG. A focus was put on web services that map to the IEC 61850 standard.

2.3.2 Purpose

This literature study focuses on leveraging IP-based IoT communication networks, including MQTT, Wi-Fi, and GSM, rather than Power Line Communication (PLC). While IEC 61850 is traditionally associated with substation automation, its adaptation to IP-based smart metering environments ensures standardized and structured communication between devices. Unlike PLC, which relies on electrical wiring for data transmission, IoT-based networks allow for greater flexibility, scalability, and integration with cloud computing.

Web services can share their entire server contents with its network, or just a port to connect to various applications. A web API is a form of a web service where the API dictates the rules of communication. Using an API, the creator of the API sets rules in the form of a request model and response model. An API in basic terms provides an interface to clients to allow effective communication. The client never knows what is going on in the inner workings of the API (Coles, 2021) .

The purpose of APIs is to interact with an application, either actioning an event or sharing or storing information.

2.3.3 SOAP vs REST

There are various Web API technologies, but not all of them are using the HTTP (Hyper Text Transfer Protocol). CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Component Object Model) are some of the first web APIs that do not make use of the HTTP protocol. SOAP (Simple Object Access Protocol) and REST APIs can be used over the Internet since they inherit the HTTP protocol.

SOAP was the first API that enabled the communication between different operating systems. Making use of HTTP, communication could be achieved between different operating systems over a network. The data that is being exchanged is done so in the form of the XML (Extensible Markup Language). XML which is a markup language that can be created by any operating system able to manipulate text.

REST is an improved API standard on the already existing SOAP protocol. Making use of the JSON (JavaScript Object Notation), a less complex way of mapping objects compared to that of XML. REST makes use of URI (Uniformed Resource Identifier) to identify resources on the API. REST APIs use the 4 function calls of the HTTP protocol to identify the action that is requested. Those actions are namely: GET, PUT, POST and DELETE (Dingra, 2016).

A resource can be accessed or mutated using a REST API by providing a request to the URI of the API. For example, making a GET request to *http://www.example.com/meter?id=1*, retrieves the information model of the meter with the Id = 1. The same information can be manipulated using POST or PUT, making one of those requests to the exact same URI with Id=1, either creates it or update its data models depending on the type of request. Making a DELETE request to the URL mentioned above removes the data allocated to Id 1.

2.3.4 Breakdown of literature on Web services integration with the IEC 61850 standard

There have been many papers written based on the combination of keywords: web services and SG. The search criteria is narrowed by using the keywords IEC 61850 and web services. A couple of authors created mappings to the IEC 61850 standard by making use of mappings from an interface. Others focussed on creating models the IEC 61850 standard uses to integrate with a new system.

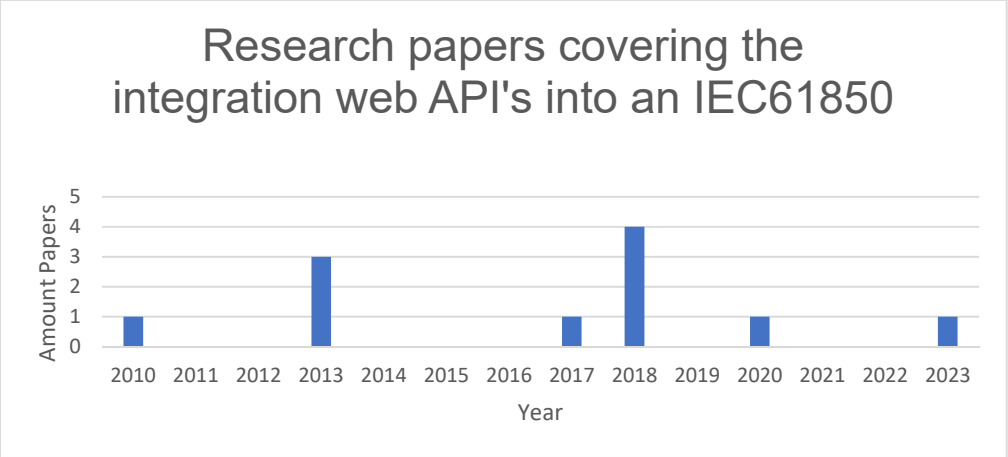

Figure 2.9: Spread of papers related to integrating of the IEC 61850 standard and web services

The figure above (Figure 2.9) shows the spread of papers from 2010 up until 2023 based on the specified research criteria. The keywords used are web services, IEC 61850 standard, and IEC 61850 API. From the below table there are some prominent authors regarding the mapping of the IEC 61850 standard to the standard of CoAP (Constrained Application Protocol). Iglesias-Urkia, et al., (2018), not only mapped the IEC 61850 standard to the CoAP protocol the researchers also integrated an IEC 61850 system with the protocol. The researchers also compared it to other API protocol mappings with the focus on performance. All the above-mentioned research was conducted in 2018 creating a spike of research in web service and IEC 61850 key in that year.

Table 2.7: Papers covering IEC 61850 mappings to web servers

Papers	Mapping to IEC 61850 used.	Use of IEC 61850
(Pedersen, et al., 2010)	REST-to-IEC 61850	Creating a mapping from REST-to-IEC 61850
(Karnouskos, et al., 2013)	Communication standard not specified.	
Research on IEC 61850 Gateway based on MMS mapping (Hou, et al., 2013)	MMS	Creating a web service gateway for MMS to act as intermediary between a non-IEC 61850 component and a IEC6150 Component
Integration of IEC 61850 MMS and LTE to support smart metering communication (Pham, 2013)	MMS	Creating an MMS gateway using LTE technology to submit metering data to a MDMS over a large geographical area.
(Yun, et al., 2017)	REST-to-IEC 61850	
Internet of Things Integration in SG(Pramudhita, et al., 2018)	None	Review IoT in SGs and creates a layered approach to incorporating IoT in SG. Referring to Web services.
IEC 61850 meets CoAP: Towards the integration of Smart Grids and IoT standards (Iglesias-Urkia, et al., 2017)	CoAP-to-IEC 61850	Creating a mapping between CoAP and IEC 61850
Integrating Electrical Substations within the IoT using IEC 61850, CoAP and CBOR (Iglesias-Urkia, et al., 2018)	CoAP-to-IEC 61850	Using a mapping between CoAP and IEC 61850 to integrate IoT into the power grid
Validation of a CoAP to IEC 61850 Mapping and Benchmarking vs HTTP-REST and WS-SOAP (Iglesias-Urkia, et al., 2018)	CoAP-to-IEC 61850 REST-to-IEC 61850 SOAP-to-IEC 61850	Comparing speeds of different IEC 61850 mappings to determine which one is the best.
Integration of IoT Technologies into the SG(Cavalieri, et al., 2022)	REST-to-IEC 61850	Creating a system that incorporates IEC 61850 with a MQTT services that is connected to a REST-API to communicate events with a server.
Design and implementation of multiprotocol framework for residential prosumer incorporation in flexibility markets (Gayo-Abeleira, et al., 2023).	MMS	The author integrates IEC 61850 with IoT technologies to create a prosumer which integrates with the power grid.

2.3.5 Comparative Analysis of web architecture design in the SG.

All software solutions require an architecture to determine how the business requirements are fulfilled. Each type of business has an architecture associated to its code business requirements. For example, the power grid requires a different architecture of that of a bank. This section discusses the papers in the above table which elucidate the architecture of web services in the IEC 61850 compliant power grid.

Yun, et al (2017), created a framework for the use of APIs to monitor the conditions of power equipment in the power grid. Using techniques used in the field of the IoT (Internet of Things), the researchers create a RESTful API by making use of ASP .NET Web API.

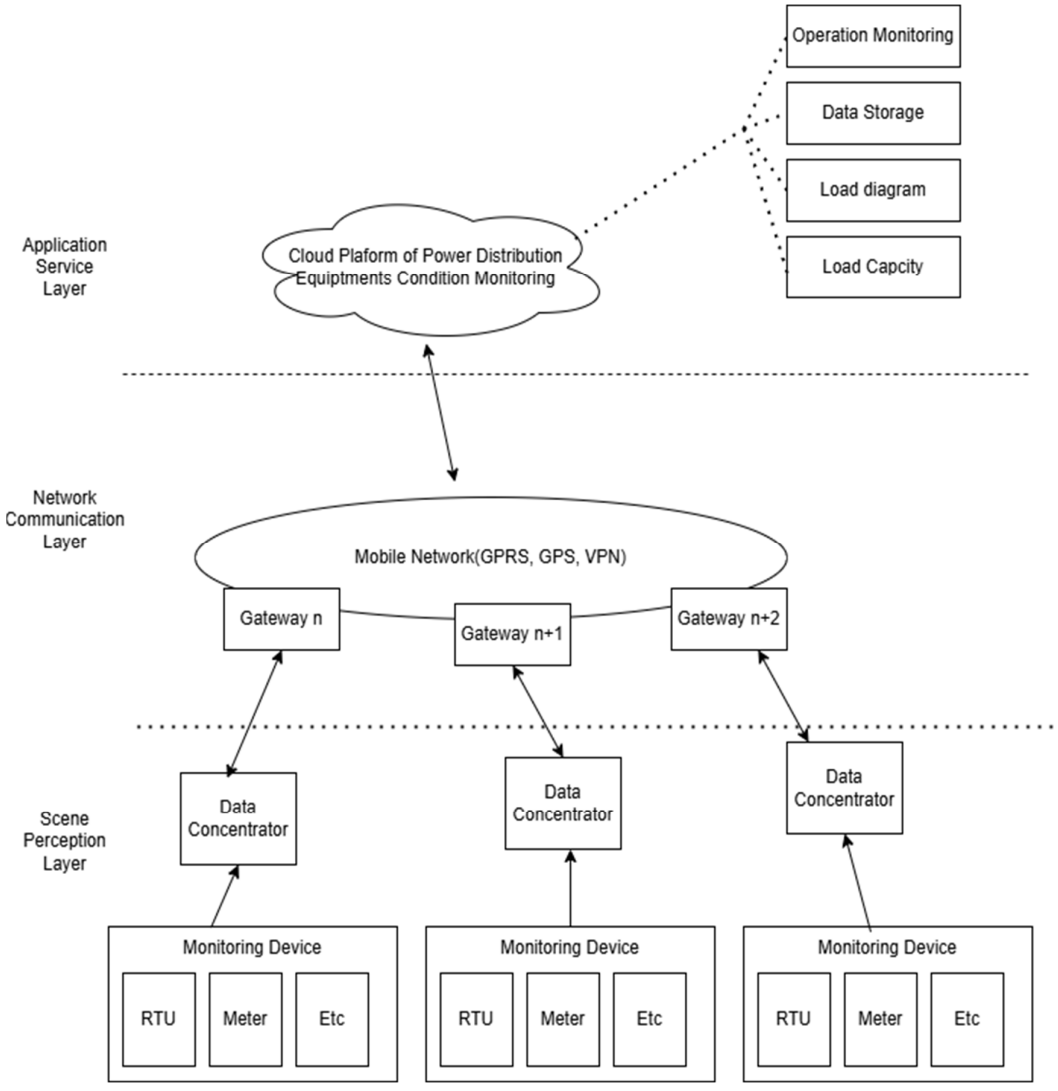

Figure 2.10: Cloud Platform Layers of power equipment monitoring (Yun, et al., 2017).

Figure 2.10 shows the layers of the framework the researchers created. It can be seen from the above figure that they hosted their application on a VPN (Virtual Private Network) network using GPRS (General Package Radio Service). The VPN is used to segment their network and allow encryption between points to make the transmission of data more secure.

Making use of a RESTful API the monitoring devices can communicate with API using URI's. The monitoring devices in the *Scene Perception Layer* are using meters and multi-function sensors to collect data from the power distribution equipment. Using a GPRS gateway all the power equipment readings are converged to a single point using RS-485 communication. All the data is then *Packaged* by the gateway and sent to the API via a URI and the GPRS wireless network.

In addition to the above they created a user platform to view/control all the above data and power equipment. This platform had multiple uses mainly:

- SCADA Platform
- Assets management platform
- Operation management platform
- Energy management platform
- Status maintenance platform
- Data visualization platform
- Big data analysis platform
- User management platform

All the above functions are built around the view/control of the power system data to allow various uses cases regarding power distribution to be fulfilled. The application was built to accommodate electrical engineers on any device with a browser to have access to the resources of a distributed energy network.

Pramudhita, et al (2018) discussed the integration of two standard IoT data collection/sensing integrations in the SG.

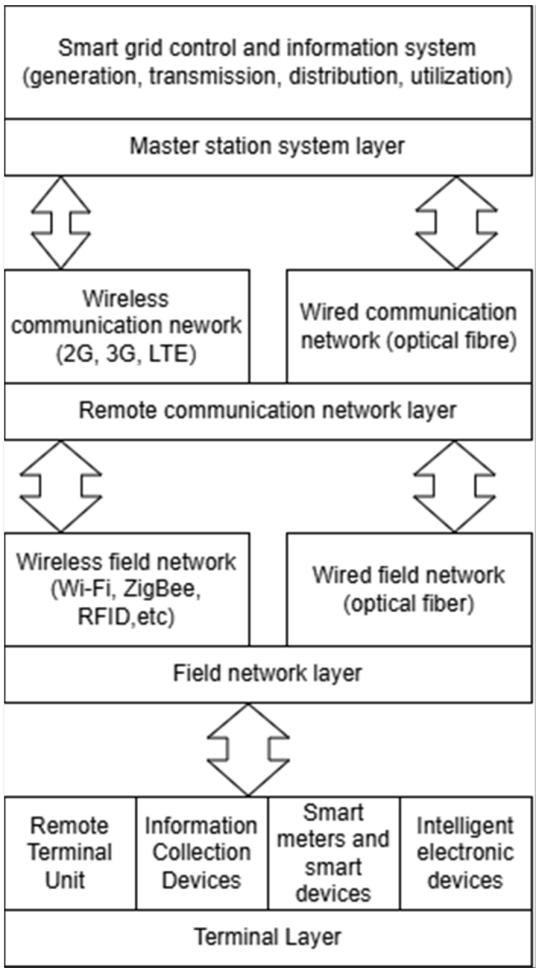

Figure 2.11: IoT abstracted layer architecture (Pramudhita, et al., 2018)

Figure 2.11 above shows the layout of the IoT architecture to enable data gathering from low level a device such as SMs, IEDs, RTUs and various devices that would sit at a metering/control level in a substation. The various layers abstracts what each layer would provide in a power system looking to control/monitor various power reading/controlling equipment.

The first layer is the terminal layer, where all the power distribution equipment would be located. Generally, this type of equipment sits on top of the control level in a substation.

The field network layer is a local network layer to connect and group all power equipment in a certain area. This type of network does not span over large geographical distances. This network is used to group a certain substation/residence or even a neighbourhood.

The remote communication network layer connects all the different field networks into one large network by means of data concentrators. These networks can stretch over

large geographical distances. They networks include but are not limited to: GPRS, 2G, 3G, LTE, 4G, WiMAX.

The top layer of the architecture, the master station system layer is the Web Service that absorbs and uses the information to control the logic of all the system. This layer is normally allocated to a control centre or wide area management system (WAMS). In a SCADA control centre, the information supplied by this layer would be extracted to allow for grid visualization.

Pedersen, et al (2010) created a mapping from the IEC 61850 standard to a RESTful API. Using the distinct rule in RESTful APIs they made use of the unique URIs in REST that mapped to resource identifiers in the IEC 61850 standard. For example: CHP1/MMXN1.Watt.mag.f is a reference has a logical device named CHP1, which has logical node MMXN1. MMXN1 has Data Object Watt. Watt has a data attribute mag, which contains data f. He mapped this path to rest by using a URI path. The REST URI for this same resource can be retrieved using:

http://iec61850webplatform.com/CHP1/MMXN1/Watt/mag/f. Making use of POST and PUT the value of the reference *f* can be manipulated.

2.3.6 Comparative Analysis of web service IEC 61850 mappings

Table 2.8 shows how the identifier IEC 61850 is converted into the resource identifiers RESTful APIs used to identify functionalities.

Table 2.8: ACSI TO REST MAPPING (Pedersen, et al., 2010)

Url	Meth.	ACSI equivalent
/	GET	GetServerDirectory (GetAllDataValues) (GetDataValues)
/[LD]/	GET	GetLDDirectory (GetAllDataValues) (GetDataValues)
/[LD]/[LN]	GET	GetLNDirectory (GetAllDataValues) (GetDataValues)
/[LD]/[LN]/[DO]	GET	GetDataDirectory GetDataDefinition (GetAllDataValues) (GetDataValues)
/[LD]/[LN]/[DO]/[DA]	GET	(GetAllDataValues) (GetDataValues)
/[LD]/[LN]/[DO]/[DA]	POST	SetDataValues

The above research work shows how interoperability can be achieved by mapping REST URI paths to that of the IEC 61850 standard. While the IEC 61850 standard has its communication protocols that are very reliable in the substation. The REST mapping allows for easy integration with WAMS, SCADA and other DERs (Distributed Energy resources). REST is widely used by all industries to play the role of the back-end of a computer system (Coles, 2021).

Iglesias-Urkia, et al (2018) creates a framework for mapping the IEC 61850 ACSI services to that of CoAP. The performance of each to-IEC 61850 mapping is measured using SOAP, REST and CoAP (Constrained Application Protocol) in an IEC 61850 environment. Using the mappings created by other researchers namely Pedersen, et al., (2010), which created a REST-to-IEC61850 framework.

CoAP is a protocol created to help IoT devices that are constricted by a non-functional issue, for example: memory, network band width, little ROM, or little RAM. The protocol is a REST-like protocol which support the use of HTTP methods, without using the HTTP protocol. Where HTTP is a TCP based protocol, CoAP is a UDP based protocol. It is a lightweight replacement with only a 4 Byte header.

In addition to creating a mapping for CoAP to IEC 61850 MMS, the author created a framework to allow a no-payload response and the observer design pattern. The no-payload option tells the server that the client does not need the resource representation. The observer design pattern is used to subscribe to an endpoint. Which is like that of MQTT. In this case the client would make an API call indicating to the server that it would like to continuously make a call to the address mentioned in the call. Figure 2.12 shows the sequence of interaction between the server and client. The server requires an endpoint and actions. In the example, the initial GET has request “obs+no-payload”, which tells the server to subscribe the client to this endpoint with no payload being returned.

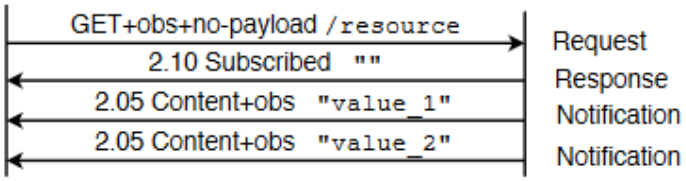

Figure 2.12: Interaction between observer-client and server (Iglesias-Urkia, et al., 2018).

A similar method to that of Pedersen, et al (2010) is used to map CoAP to that of ACSI in the IEC 61850 standard. Each ACSI service is mapped to a resource of the CoAP URI, making use of parameters in the URI, each service and resource can be identified.

Function	URI	Method	Id	Notes
Basic				
Server				
GetServerDirectory	coap://{host}/LDs	GET	1	Logical Devices
GetServerDirectory	coap://{host}/Files	GET	2	Files
Logical Device				
GetLogicalDeviceDirectory	coap://{host}/LDs/{LD}	GET	3	
Logical Node				
GetLogicalNodeDirectory	coap://{host}/LDs/{LDs}/{LN}/{ACSIClass}	GET	4 5 6 7	Datas Datasets BRCBs URCBs
GetAllDataValues	coap://{host}/LDs/{LDs}/{LN}/AllValues[?FC={fc}]	GET	8 9 10	Big payload Medium payload Small payload
Data				
GetDataValues	coap://{host}/LDs/{LDs}/{LN}/Datas/{Data}	GET	11	
SetDataValues	coap://{host}/LDs/{LDs}/{LN}/Datas/{Data}	PUT	12	
GetDataDirectory	coap://{host}/LDs/{LDs}/{LN}/Datas/{Data}/Directory	GET	13	
GetDataDefinition	coap://{host}/LDs/{LDs}/{LN}/Datas/{Data}/Definition	GET	14	
Dataset				
GetDataSetValues	coap://{host}/LDs/{LDs}/{LN}/Datasets/{Dataset}	GET	15	
GetDataSetDirectory	coap://{host}/LDs/{LDs}/{LN}/Datasets/{Dataset}/Directory	GET	16	
Reporting				
BRCB				
Report	coap://{host}/LDs/{LDs}/{LN}/{BRCBs}/BRCB/Reports	Notification	17 18	Big payload Small payload
GetBRCBValues	coap://{host}/LDs/{LDs}/{LN}/{BRCBs}/BRCB	GET	19	
SetBRCBValues	coap://{host}/LDs/{LDs}/{LN}/{BRCBs}/BRCB	PUT	20	

Figure 2.13: CoAP to ACSI function mapping (Iglesias-Urkia, et al., 2018)

Figure 2.13 shows an example of the mapping URI to the various ACSI functions. The functions are also dependent on the CoAP Method that is being utilized. *GetDataValues* and *SetDataValues* have the same URI, but their distinct methods identify the action that should take place between either setting the value or getting the value.

Using this mapping of CoAP-to-IEC 61850, the performance of it was compared to that of REST-to-IEC 61850 and SOAP-to-IEC 61850.

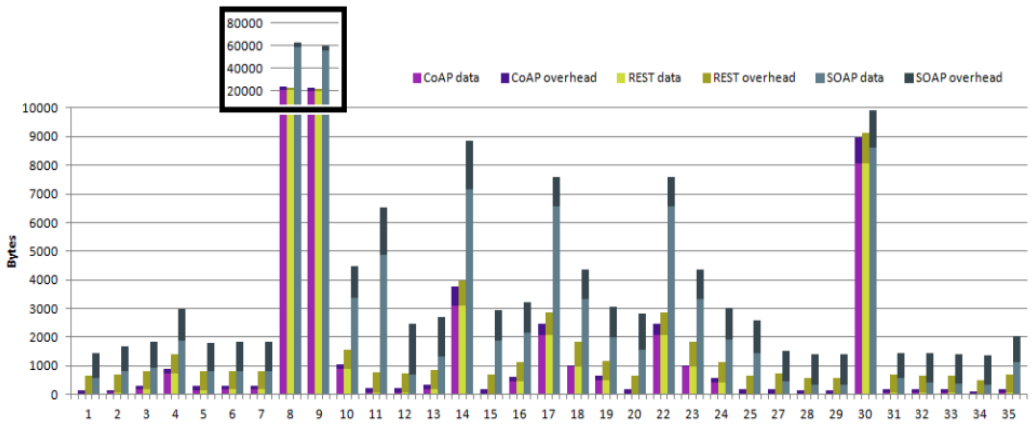

Figure 2.14: Data and overhead bytes (Iglesias-Urkia, et al., 2018)

Figure 2.14 shows the results of the comparison between the 3 mapping performances. In all cases the SOAP has a higher data and overhead exchange than the other 2 mappings. The figure also shows, the bigger the data is being sent via CoAP, the more overhead it creates. At a point when the data payload becomes larger, the REST mapping becomes more efficient with its overhead. In the figure once the payload reaches 20 kilo bytes, REST starts performing better than the CoAP mapping. Although the CoAP technology is still relatively new, there are many improvements the researcher can make to allow the CoAP to become more efficient.

2.3.7 Discussion

This section documents a literature review to determine the technologies, frameworks and architectures a web API would have in an IEC 61850 environment.

Yun, et al., (2017) and Pramudhita, et al., (2018) created a very similar framework to build an architecture, creating a hierarchy of all the components that are used in a SG environment. This hierarchy consists of the various components a grid needs with a web server at the top of the hierarchy. Pramudhita, et al., (2018)'s focus is towards the consumption of metering data and using IoT based standards to have all metering devices use an API architecture. Yun, et al., (2017)'s framework consists around the overall control of the power grid using a SCADA system.

The IEC 61850 standard was found to be mapped to various other communication protocols to allow integration with other systems. Pedersen, et al., (2010), created a mapping for REST from the IEC 61850 standard. REST which was at the time the newest framework of API communication (Coles, 2021). Iglesias-Urkia, et al., (2018), created a mapping to a very new IoT REST standard that is perfect for low power devices. They mapped CoAP to the IEC 61850 standard. In addition, they also created an implementation and compared the results to that of the other mappings that maps to the IEC 61850 standard-based implementation.

The following section provides the literature review and discussion on the SM design.

2.4 Smart Meter Design

2.4.1 Introduction

The SM forms the central part of this proposed research work. The SM is the evolved version of the conventional power meter. The main purpose of the power meter is to determine how much power a household uses and report that to the power utility to appropriately bill the owners on their power usage. Since the invention of the conventional power meter was invented the needs of the power grid have evolved.

Some of the new needs of the SM can be addressed by creating a newer meter, a smarter meter.

This section documents a literature review to determine what the requirements are of the SM and how those needs fit into the bigger picture i.e., the SG. The goals of this literature review can be viewed as follows:

- Determine what a SM is.
- Determine what capabilities are required of the SM.
- Determine if there are any standards regarding the SM.
- Investigate design patterns of the SM.

2.4.2 Purpose

A SM is a power meter to help integrate every load within the SG. The SM is the coordinator between the load and supply. Determining what the grid requires is an important step in creating a SM. The way that power is being used might differ from country, culture and lifestyles. These factors change the requirements of a SM. Using a generic South African home as a reference it is determined what is required for the SM (Das, et al., 2024).

2.4.3 Breakdown of literature

Table 2.9: Components used by different authors to construct a SM

Author	Device Level	Central Processor	Power Reading	Communication	Power Switch	Extra Components
(Burunkaya & Pars, 2017)	Wireless Sensor network	STM32 Nucleo-64 board	Current Transformer and a Voltage transformer	ZigBee	Relay	N/A
(Hlaing, 2017)	Wireless sensor network	Arduino Leonard Pro Macro	Current – AC712 Voltage – Zmpt101b	Esp8266 external Wi-Fi module	N/A	N/A
(Dalpiaz, et al., 2018)	Mains	STM32L4	Current Sensor	RFM95W – RF radio	N/A	N/A
(Sayed, et al., 2019)	Mains	Arduino Uno	Current Transducer and Voltage Transducer	ESP8266 Wi-Fi	Arduino Relay shield	N/A

Author	Device Level	Central Processor	Power Reading	Communication	Power Switch	Extra Components
(Jasim & Abdalla, 2020)	Mains	ESP32	PZEM-004T	WI-FI	Relay	RFID – Payment Buzzer - Alarm
(Hasam & Kadhim, 2020)	Mains	Arduino Mega	PZEM-004T	MQTT + GSM	Relay	Storage – SD Card Reader
(Midul, et al., 2023)	Mains	Arduino Nano	Not specified	GSM Short Message Service (SMS) ESP-8266 Wi-Fi Module	N/A	N/A

Table 2.9 contains a list of components that are used to create SMs from various authors. All the components have a very distinct role in the creation of the SMs. The above table investigating all research work have the following type of components in common.

2.4.4 Comparative analysis of Smart Meter design

Hlaing (2017)'s goal was to create a wireless sensor network of SMs to gather data on appliances, to inform people of their power usage at home. The main purpose of this research work was to introduce a cost-effective way of installing a SM to efficiently communicate with an online service via Wi-Fi. The ESP8266 Wi-Fi module by Espressif is the module being introduced. The benefit of this module is that it is low power, has its own processing and storage.

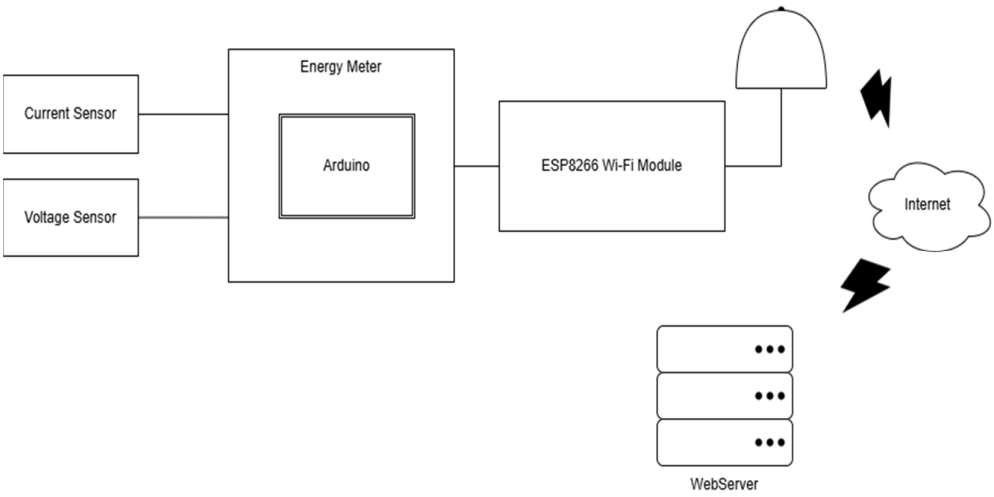

Figure 2.15: Block diagram of Smart Meter abstract components (Hlaing, 2017).

From Figure 2.15, Hlaing (2017)'s version of a SM is to acquire the voltage and current data. The voltage and current are then processed on the Arduino (Central processing unit) and then submitted via an Internet connection made available by a WIFI connection and the ESP 8266 device. If the WI-FI is connected to the Internet, then the values collected from the sensor are submitted to the webserver.

Jasim and Abdalla (2020), created an electricity payment metering system, where the user must pay for credit which is associated to their RFID (Radio Frequency Identification) card to allow the meter to switch the power on for a duration of time. The system uses a central online server to keep track of RFID cards, payments, and users. The system contains three devices. The SM, the central unit, and the paying device. The payment device consists of a micro controller and an RFID reading/writing unit. The payment device writes the RFID serial number along with device details to the central unit via an online API, hosted on the central unit. The data is stored along with various user data to make the link between user, card and SM.

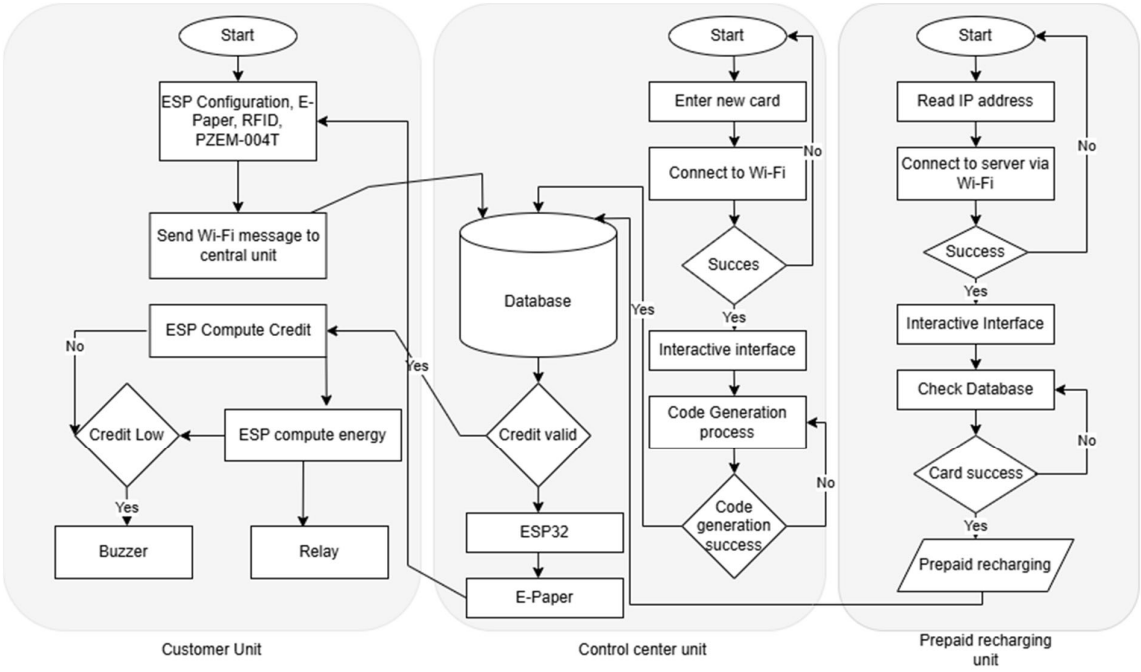

Figure 2.16: Flow chart of SM payment system (Sayed, et al., 2019)

Figure 2.16 shows a flow chart of all three parts of the system being integrated. All three parts of the system has at some point communication with the database. The customer unit (SM) checks if there is credit available. Only if there are credits available, will the relay of the customer unit switch on. The control centre has its own interface to add new clients and SMs to the system. The flowcharts are dissected below according to their function.

[CUSTOMER Unit/SM FLOW CHART]

1. The customer unit initialize the various components attached to it. Three modules are initialized: RFID reader, PZEM-004T power reading module and the E-Paper module.
2. The Wi-Fi module makes a request to the control centre unit

[CONTROL CENTRE FLOW CHART]

1. The control centre starts its looping process by making connection with a Wi-Fi network. This will run until a network connection Is established.
2. A User interface is then displayed to either add a new user or upload credits for the use of electricity.
3. All action is then communicated with the database.

[PREPAID RECHARGING UNIT]

- The recharging unit initiates connection with the Wi-Fi network, this step will be repeated until a Wi-Fi connection is established.
- A user interface is then displayed.
- The RFID card is then used by the customer to identify themselves.
- The prepaid recharging unit gets all the information from the database.

The power metering method is the main part of the meter, which incorporates both voltage and current measurement. The PZEM-004T multi meter module incorporates a hall effect current transformer to measure current and voltage transformer. Using these two values a library is used to calculate the power usages. The hall effect current transformer allows current to read without intruding on the circuit; using the magnetic fields to create a current in the current sensor (Dewi, et al., 2016).

Hasam and Kadhimd (2020) also created a similar system to Jasim and Abdalla (2020) although their components may be different, the same goal was achieved. Hasam and Kadhim (2020) also created a server which consisted of a Raspberry Pi and a MySQL Database to store data. Using an Arduino Mega along with various other components a device that resembles a SM was created. Each SM subscribes to the server (Raspberry Pi's) MQTT service. All the data is then logged and submitted to a database to be analysed.

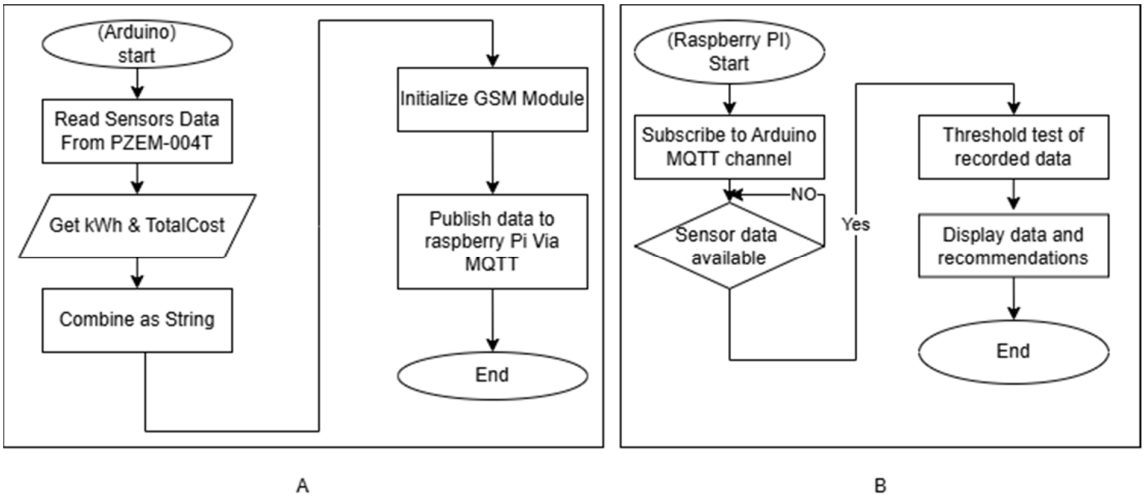

Figure 2.17: Flow charts of Raspberry Pi Meter reading and displaying data (Jasim & Abdalla, 2020)

Figure 2.17 depicts a flow chart which describes the logical flow in which the SM (A) submits data to the MQTT server (B). The two flow charts are dissected below.

[FLOWCHART A]

Flow chart A in the figure above depicts these steps:

1. Once the Arduino starts it will read sensor data, taking readings from the PZEM-004T module, and storing the data.
2. The Arduino module using stored procedures determines what are the cost of power currently being used.
3. Combine all the above gathered and calculated information to create a payload of information.
4. The (Global System for Mobile Communication) GSM mobile network module is initialized with the payload.
5. The data is then published to a MQTT broker using the newly established GSM connection.

[FLOWCHART B]

1. The Raspberry PI is started and is immediately subscribed via the broker to a topic where all reading data will be published.
2. The Raspberry Pi queries whether there is data in the queue reading for consumption. If there is nothing to consume the Raspberry Pi will repeat this step.
3. If the MQTT broker does have data available to be consumed. The Raspberry Pi will do various tests on the information to determine if it is correctly standardized.
4. The information will then be consumed by an HMI and Node-red GUI.

Dalpiaz .et al (2018), created a SM capable of harvesting power from the line it is monitoring. The SM created is non-intrusive, battery free and has low power consumption. Avoiding battery related SMs reduces the SM maintenance. The non-intrusiveness allows the installation and maintenance to be easier. Low power consumption prevents the meter from influencing the power readings of the meter itself.

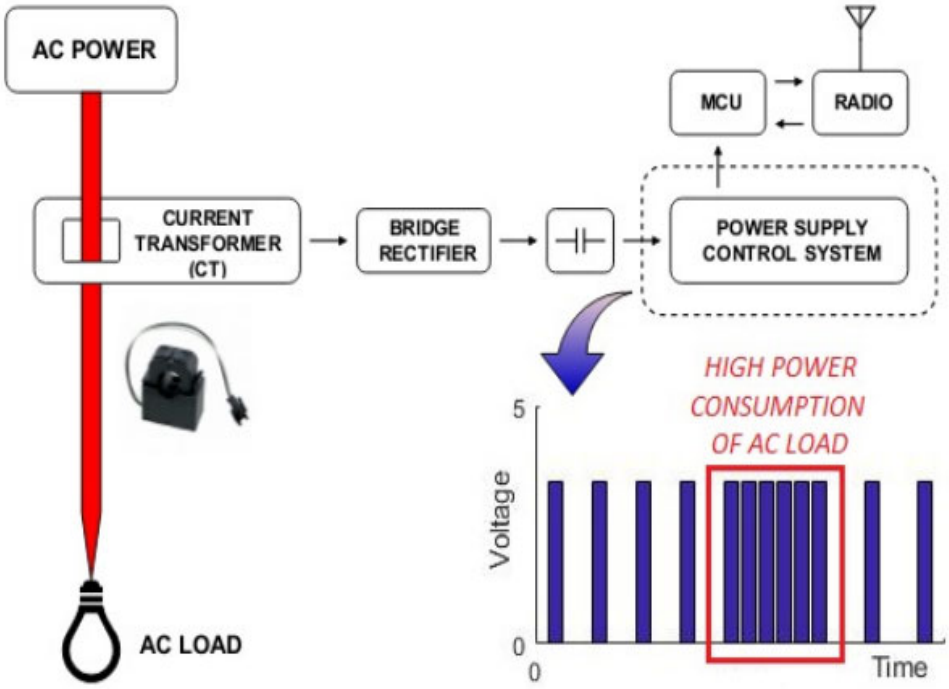

Figure 2.18: Basic Functioning of SM (Dalpiaz, et al., 2018)

The low power consumption goal is achieved by only activating the Microcontroller Unit (MCU) when a transmission is required, otherwise the MCU remains in sleep mode. In Figure 2.18 the current transformer is connected around the line that needs to be monitored. The current is fed to a bridge rectifier which charges a capacitor. Once charged, it sends a voltage to the MCU to activate it and send a signal via the radio. The bigger the load consumed, the faster the capacitor will charge and send out a signal. The capacitor being of a certain size, would always release a charge over a certain amount of time. Discussion

All authors had a very similar approach to creating a SM. There are four common roles in SM construction that needs to be built in order to qualify the meter as a SM. Below is a list of the various component roles:

- Central Processing
- Power Reading Module
- Communication
- Switch

Each component has its own distinct role in a SM. The Central Processing Unit (CPU) functioning as the common connection component and where logic and interface live.

The power reading module is the module that is used to transduce/convert current into a readable form for the CPU.

The communication module can take the form of any network media interface. Wireless networks are the most common as wired connections seem to create changes for the metering network. Jasim and Abdalla (2020), makes use of a Wi-Fi network using the ESP32 module, enabling a Wi-Fi network-based system. (Hasam & Kadhim, 2020) use a very similar approach to allow locally based meters to communicate with a centralized local server, before transmitting it to a high-level server using GSM. (Midul, et al., 2023), uses the same approach as (Hasam & Kadhim, 2020) making use of Wi-Fi to allow local devices to communicate via Wi-Fi and reporting services are communicated using large mobile networks via GSM, 3G or 4G.

Although it seems that Wi-Fi is the best communication media SMs use to communicate locally, (Dalpiaz, et al., 2018), made use of radio signals and a low powered microcontroller to communicate all power readings via a frequency-based communication. The higher the frequency of power signals sent, the higher the power usage. This approach is the most different approach of power metering, all other authors made some use of ASCII-based communication protocols.

SM's are built to measure the power of either a wireless sensor network where home automation devices and IoT protocols are integrated within the home. These SMs are not built to control the load, but to measure the power as a wireless sensor network. SMs are built to measure the main power source, and from what was identified in the research, can control the load with the help of a switch.

The following section presents the concluding remarks to this chapter.

2.5 Conclusion

The literature review provided a comprehensive analysis of existing research on smart metering systems, communication protocols, and the IEC 61850 standard. Key gaps were identified, particularly the lack of interoperability in distributed energy systems and the challenges posed by proprietary communication frameworks. The analysis underscored the critical need for standardized communication to ensure seamless integration of devices across heterogeneous networks.

The discussion on web services highlighted their pivotal role in enabling efficient and scalable communication within SG systems. The integration of web service architectures, such as REST APIs, with IEC 61850 MMS protocols was shown to offer significant advantages in terms of accessibility, modularity, and performance. These

web service frameworks allow for easier adaptation to varying network conditions and enable flexible data exchange, which is crucial for modern SG applications.

In conclusion, the design of SMs plays a pivotal role in advancing modern energy management systems by integrating key components such as power measurement units, communication interfaces, and processing capabilities. These designs emphasize modularity, scalability, and cost-effectiveness, ensuring compatibility with existing grid infrastructure. While significant progress has been made in leveraging open-source technologies and IoT-based communication, challenges such as real-time data processing, robust security, and seamless integration with global standards like IEC 61850 remain areas for further exploration.

Similarly, the MMS protocol and IEC 61850 standard have been shown to be essential in achieving interoperability and standardization within the SG. The object-oriented approach of MMS facilitates efficient real-time data exchange, while the scalable and hierarchical architecture of IEC 61850 ensures seamless integration of diverse devices. Together, they offer a robust framework for addressing communication inefficiencies and interoperability challenges. However, their application in residential and distributed environments highlights the need for continued research to enhance scalability, address security concerns, and unlock their full potential for widespread adoption.

Chapter 3 describes the methodology and the creation of a plan together with the introduction of various software and hardware models to create the proposed system.

CHAPTER 3

RESEARCH METHODOLOGY AND PLANNING PHASE

3.1 Introduction

This research work is in the majority a software-based project, where various systems are integrated to achieve a common goal. In the SDLC (Software Development Life Cycle), a planning phase must commence before an implementation phase can happen. This chapter documents the planning phase of this research work. In the SDLC planning phase some units (components of a system) may need to be completed partially to ensure that certain integration points are viable.

The planning of an Application Programming Interface (API) is crucial in ensuring interoperability and data exchange between various components of the smart metering system. By designing a robust and standardized API, utilities can streamline the integration process, making it easier to incorporate new technologies and services in the future. This promotes innovation and allows for the development of more efficient and customer-centered solutions.

Moreover, the integration of the IEC61850 MMS (Manufacturing Message Specification) server into SMs is a fundamental step toward achieving seamless communication within the electrical grid. This standard ensures that the devices and systems involved in energy distribution can communicate effectively, allowing for coordinated operations and improved grid reliability. By adhering to the IEC 61850 standard, utilities can reduce downtime, respond to outages more quickly, and optimize grid performance.

The MMS server and the Smart Meter API both receive input based on what the Smart Meter (SM) provides. Therefore, the section which plans the server components are both dependent on the inner workings of the SM. This dependency resolution determines the flow of this chapter.

This chapter has 3 planning phases, which are divided into their own distinct sections. Section 3.2 contains the partial planning and implementation of the SM, which other sections require to correctly plan the integration points. Section 3.3 outlines the various API (Application Programming Interface) components, and the planning and the construction thereof. Section 3.4., plans the construction of the MMS Server and the dependencies thereof. Section 3.5 concludes the chapter and provides a high-level overview of what is discussed in this chapter.

3.2 Smart Meter

There are various SMs on the market. Although the SMs are capable of great things, they are not standardized to suit the same purpose. Some meters are created to suit a niche market or used to be compatible with a certain manufacturer. The main purpose of using the IEC 61850 standard in the power grid is to standardize communications to allow for interoperability between equipment from different vendors or Original Equipment Manufacturers (OEMs). Designing a SM with the goal of interoperability allows various manufacturers to use the SMs output and input data for various applications. The application of the IEC 61850 standard in this project allows for different systems to communicate with the SM.

This section practically implements the SM with it modules, to determine what the data structure consists of. Knowing this data structure allows the other units namely the MMS server and SM API to be planned according to what data is gathered and sent to that component. The next section introduces the requirements for the SM.

3.2.1 Requirements for the SM

Using the results of the literature review, a concise list of components is selected to fulfill the functions of a SM. The SM is then abstracted to three main functions: logic, metering, and communication. Using these three functions as a guideline the components for the SM are selected.

The first component that makes a SM smart, is the brain - the CPU (Central processing unit). The CPU can take any form of device that is able to make logical decisions. The CPU can be any form of computer. This device is required to integrate different components to classify the device as a SM. The second component is the meter itself, which is an electricity sensor. The sensor should be capable of measuring an electrical attribute of the electricity usage, i.e., Current, Voltage, Power and Frequency. These values help the system in making calculations and decisions based on the data. The third component is the communication module. The communication module uses a certain specified primary network communication media. This media is based on what is available in the environment of the meter. This network is used to communicate with a centralized data server over the Internet. In addition to all the above components there needs to be some method of controlling the meter which is connected to the main AC power supply of the residence. A relay is situated intrusively to be able to control the power supply supplying the residence.

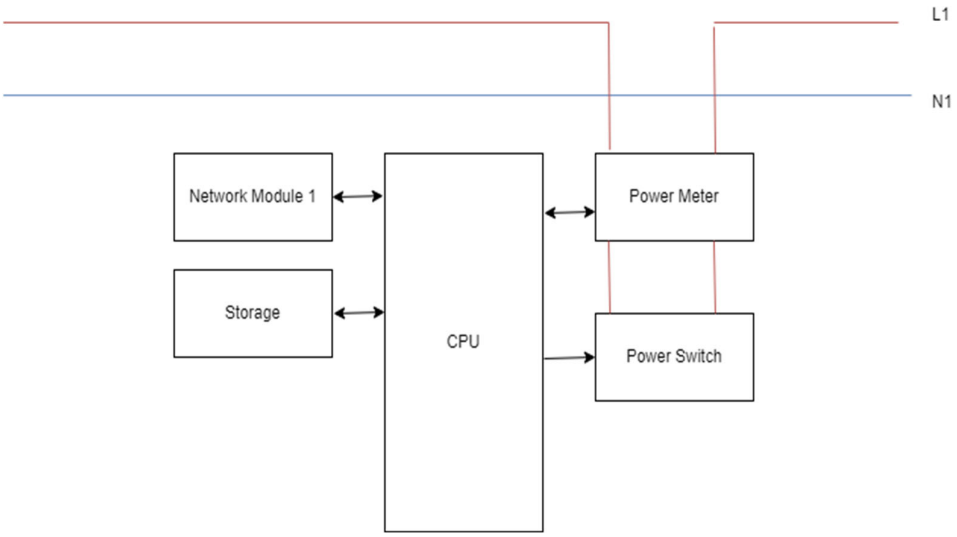

Figure 3.1 Abstraction of SM and the capabilities

Figure 3.1 shows the abstraction of how the SM is integrated with the main power supply of a household. The power meter and the power switch are connected directly to the main power supply of the household. The power meter reads the data of the power line to which it is connected. The CPU makes decisions and communicates the readings to the centralized server via a network interface connection.

Figure 3.1 shows a sequence diagram of how the main system interaction would occur between the components of the SM. At a high level, the three main parts of the system is the CPU, Power meter, and the Network interface connection.

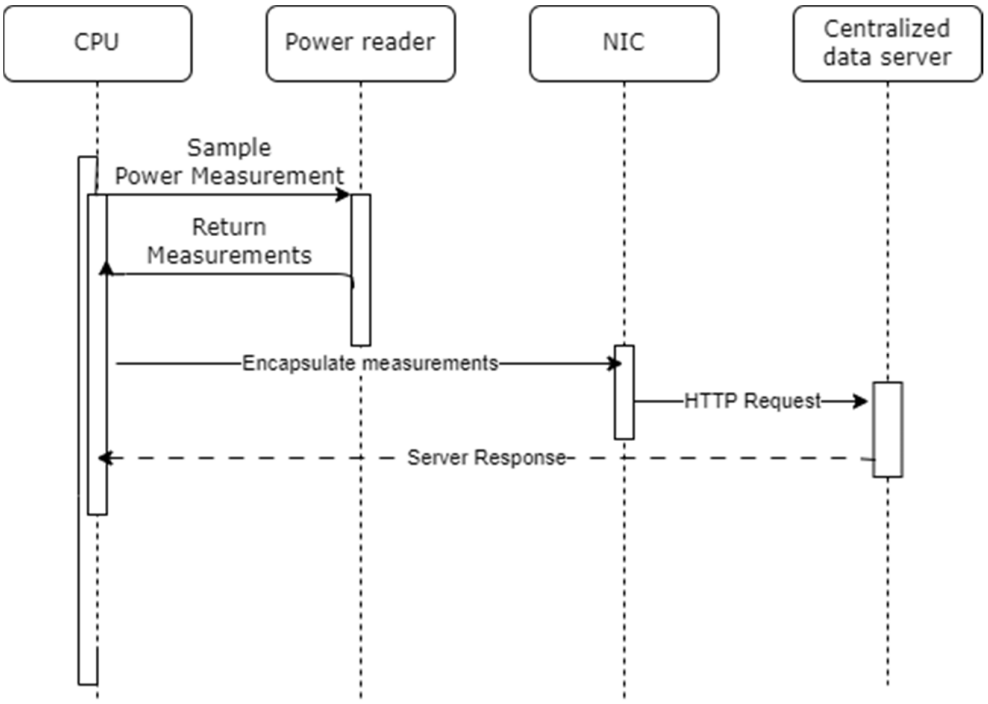

Figure 3.2: High level overview of smart meter functionality

Figure 3.2 refers to the network interface connection. This connection can be any form of info of component on the SM that allows it to communicate over the Internet. The main function of the Network Interface Card (NIC) is to communicate with the API-gateway. This is done by means of HTTP request. The following section outlines the component roles, the types and the selected components for the research work.

3.2.2 Component Selection

3.2.2.1 CPU

The CPU is the most important part of the SM, where all the components are integrated. The Raspberry PI, the SoC-like computer, is selected for its capability to integrate with more parts than a normal microcontroller. The built-in Wi-Fi and GPIO pins make it the best device to use for integration, because of the variety of software and hardware it supports. The components are integrated with the Raspberry Pi using different interfaces. Using either (Serial Peripheral Interface) SPI or the built-in GPIO of the Raspberry Pi is the best candidate for building the SM device.

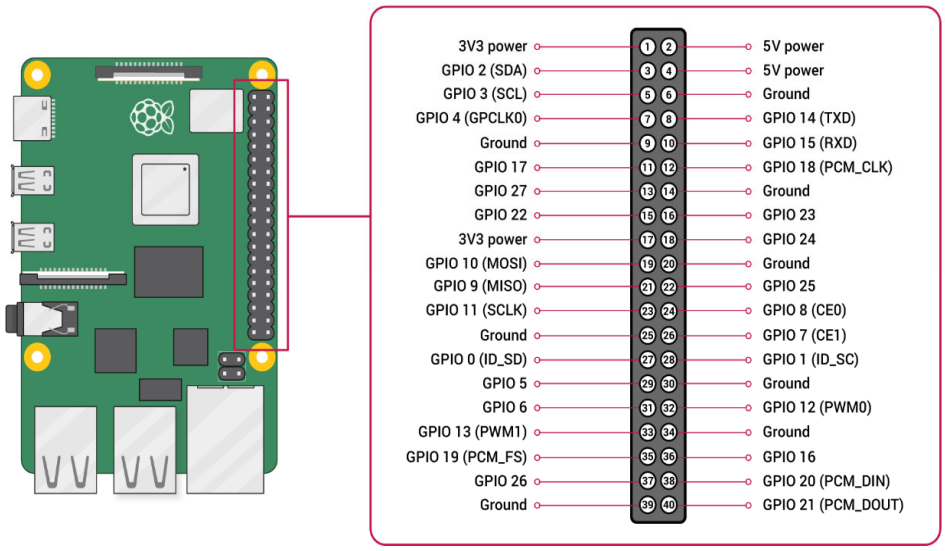

Figure 3.3 Pin Layout of Raspberry Pi (Raspberry Pi, 2023)

Figure 3.3 shows a pin layout diagram of the Raspberry PI. As mentioned, before it is selected due to its built-in Wi-Fi capabilities. The Wi-Fi module attached to the board can be seen in Figure 3.3. It can be seen from the pinout diagram, that the Raspberry Pi provides 5V power supply pins to all integrating components. In addition to the power supply pins there is a great amount of General-Purpose Input Output (GPIO) pins for input/output.

The closed hardware components which also require integration, requires another interface other than a hardware interface. The USB ports the Raspberry Pi provides creates more options of integration via SPI. The storage capabilities of the Raspberry Pi are also utilized along with its built-in operating system and file system.

3.2.2.2 Meter

The PZEM-004T multi-functional sensor module is utilized to provide metering capabilities to the SM. This component is selected due to its array of meter readings being provided. This component is an intrusive component, which is required to be connected into the circuit with an additional current transformer.

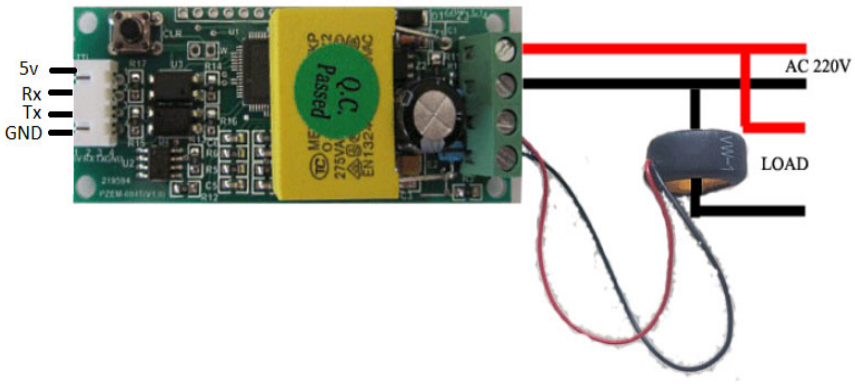

Figure 3.4: Diagram of PZEM-004T power module (nn-digital-website, 2023)

Figure 3.4 is an example of how the pin layout of the module looks. The left side of the board contains the low voltage control circuit; the right side of the board contains the high voltage reading circuit. The power reading module is connected into the load's terminal, with a current transformer connected to the load's neutral line. The series connection is to enable the power reading module to read the current.

The PZEM-004T contains a pre-coded chip that is responsible for reading the power. The device has its limitations, but the device is selected due it is limits not falling within the parameters of this project. The module can measure between 80 – 260V with 0.1V resolution and 0.5% accuracy. The module can measure up to 100A using the external transformer. In addition, the active power, power factor and frequency can be read with a 0.1 resolution, irrespective of the unit of measurement. (innovatorsgurus.com, 2023).

There are multiple ways of integrating with the module, and it is decided after integration tests that SPI is best suited due to its support on the Raspberry Pi. Another viable interface is GPIO, but there is no support for analogue data exchange on the Raspberry Pi GPIO pins.

3.2.2.3 Relay

A 5V Arduino Relay is integrated with the SM to allow the programmatic switching circuit. This relay is selected because of its simplicity. Using a GPIO pin and a 5V power pin and GND pin ensures that this relay is easy to integrate into the system. By controlling the bit, the signal pin is connected to switch the circuit on and off depending on which terminals the load is connected to. In the case of this device the load is connected to the Normally closed and Common pins.

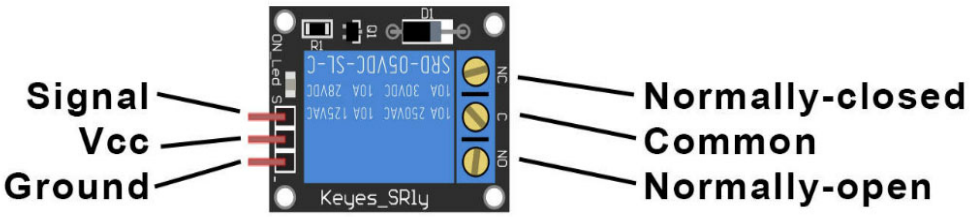

Figure 3.5: Diagram of the Arduino relay being utilized (Murray, 2023)

Figure 3.5 shows the pinout diagram of the Arduino relay. The relay as seen above is very simple to implement. It requires a power source and a digital signal to be able to be controlled. The Controlled circuit has two options of connection, normally closed (by default the circuit is on, unless power signal is given) and normally open closed (by default the circuit is off, unless power signal is given to turn it on). The SM always has the circuits powered up unless instructed otherwise, hence the normally closed configuration is selected for the SM. The following section implements all the components of the SM to determine what the output consists of. This allows a data model to be planned by the Smart Meter API and the MMS server.

3.2.3 Partial Implementation

Each of the main components that have been discussed thus far i.e. Smart Meter (CPU, Relay and Power Reading Module). The final construction of the system consists of integrating the main components:

- SM
- Smart Meter API
- MMS server

To enable these components to correctly communicate with each other, each component is partially implemented to determine if they can communicate. In the case of the system being created, the smart meter voltage/current transformer outputs data. This data needs to be interrogated to determine if the data structure, data types and the data itself is valid and usable.

By establishing what is being returned from the PZEM-004T module, it allows the implementation of the other components to be more detailed.

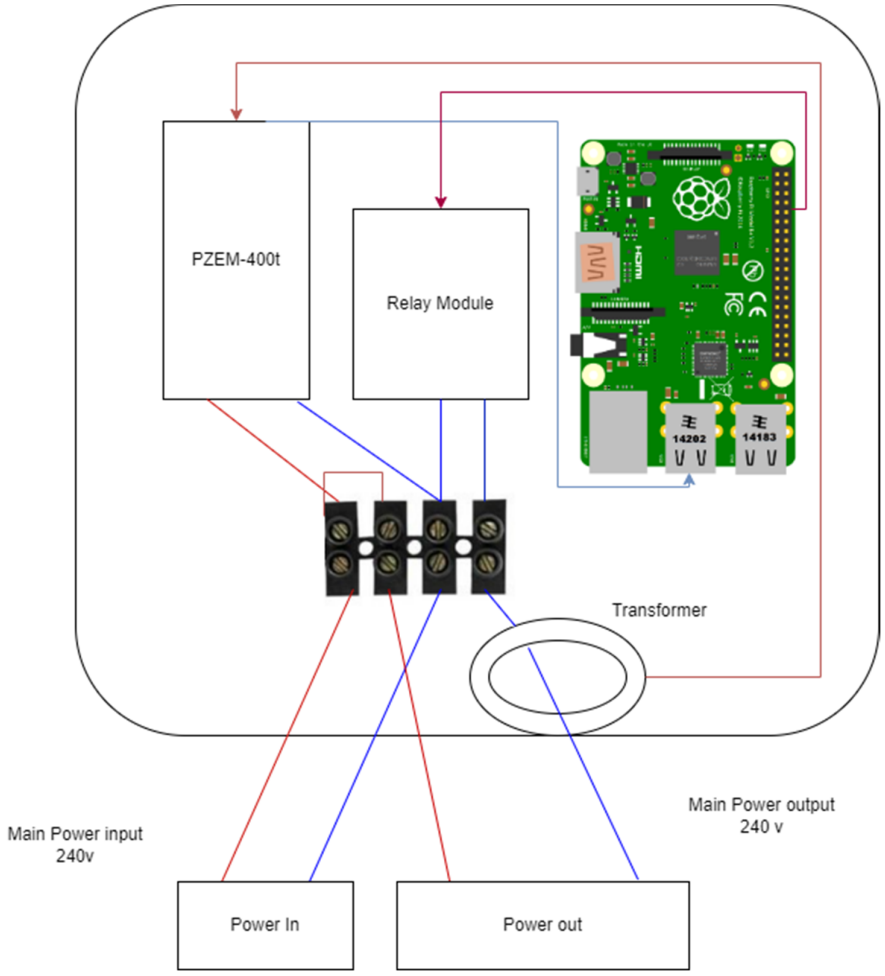

Figure 3.6: Final layout of smart meter device

Figure 3.6 shows a high-level connection of the SM device to the power connections of a main power line. It can be seen from the above figure that the PZEM-004T device has 3 connections (power in, power out and USB connection from the Raspberry PI) to allow it to properly read power. In addition, a transformer is also connected across the power out line to enable another reference for current reading. The module is connected to one of the RPI's USB3 port.


```
{
  "voltage": 229.0,
  "current_A": 0.0,
  "power_W": 0.0,
  "energy_Wh": 97492,
  "frequency_Hz": 50.0,
  "power_factor": 0.0,
  "alarm": 0
}
```


Figure 3.7: PZEM-004T data structure of a single reading

Figure 3.7 demonstrates what the power readings retrieved from the PZEM-004T consist of. Making use of a Python library named *mobus_tk* (Luc, 2023), it is possible to retrieve the values and convert it to the above JSON format. Appendix 8.1 shows the code responsible for generating the above.

With its control circuit being connected to the Raspberry Pi, the power line connection is connected as normally closed onto the relay. The Raspberry Pi sends a signal to the relay each time the power of the circuit the device is connected to, is required to be switched off. The switching signal is determined by the SM API which would provide data via an HTTP request.

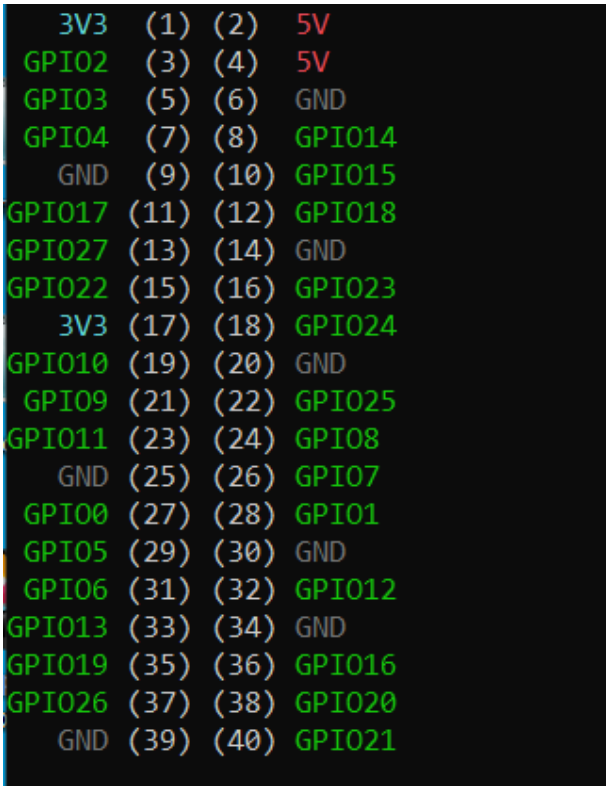

Figure 3.8: Pin labelling of the Raspberry Pi

Figure 3.8 shows the pin numbering for the Raspberry Pi. The Relay's signal pin – the control pin- is connected to GPIO 17 (physical pin 11). Appendix 8.1 shows the code that is used to test the execution of commands from the Raspberry Pi to the relay. The testing acceptance criteria is to validate communication between the Raspberry Pi and the relay. Figure 3.9 shows the signal pin of the relay is connected to GPIO17 or Physical pin 11 of the Raspberry Pi while pin 6 and pin 8 are utilized as 5V and GND, respectively. The operating circuit (240V input) is not connected for this test case.

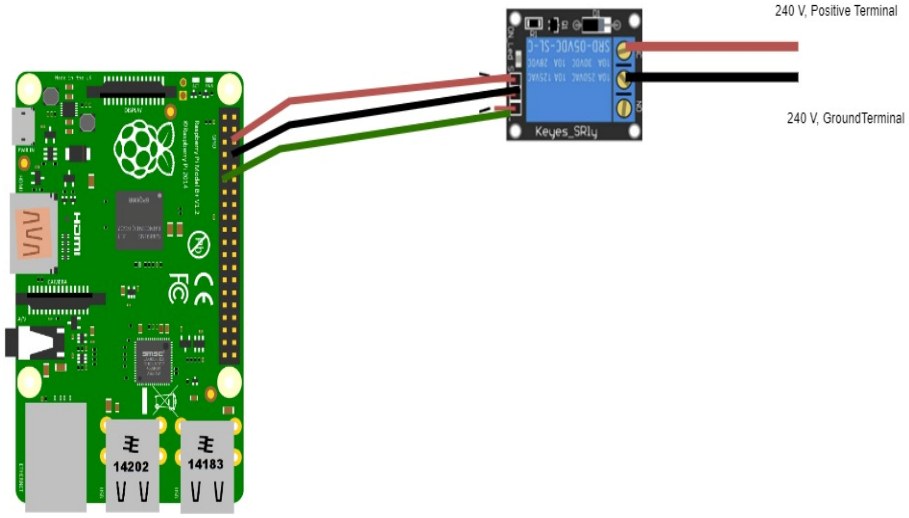

Figure 3.9: Raspberry PI Relay connection

The code in Appendix 8.1 is used to test the relay integration with the Raspberry Pi, depicted in Figure 3.9. The output of this test unit is merely to switch the relay on and off according to user input. No extensive logic has been created. At the end of the section the final integration of the three main parts of this thesis is demonstrated.

Then integrating with the already existing API endpoints on the SM using the various Python libraries already discussed, an additional Python library is added to the smart-meter client to allow it to make API calls to the SM. The code created can be found in Appendix A. Figure 3.10 shows the values that is being retrieved from the PZEM-004T module and the request being made to the SM API, which are discussed in the next section.

```
POST request sent to http://192.168.100.105:8000/Readings/f44f550a-3b08-482d-aa24-6a585286261e. Response: 200
(2310, 191, 0, 213, 0, 32878, 1, 500, 48, 0)
{
  "voltage": 231.0,
  "current": 0.191,
  "power": 21.3,
  "powerFactor": 0.48
}
POST request sent to http://192.168.100.105:8000/Readings/f44f550a-3b08-482d-aa24-6a585286261e. Response: 200
(2314, 194, 0, 212, 0, 32878, 1, 500, 47, 0)
{
  "voltage": 231.4,
  "current": 0.194,
  "power": 21.2,
  "powerFactor": 0.47
}
POST request sent to http://192.168.100.105:8000/Readings/f44f550a-3b08-482d-aa24-6a585286261e. Response: 200
(2314, 284, 0, 312, 0, 32878, 1, 500, 47, 0)
{
  "voltage": 231.4,
  "current": 0.284,
  "power": 31.2,
  "powerFactor": 0.47
}
```

Figure 3.10: Command line output from SM Python script

Figure 3.10 shows the output of the SM Python script. The figure shows that a post request is being made to: *http://192.168.100.105:8000/Readings/f44f550a-3b08-482d-aa24-6a585286261e*. This URL is pointing to *192.168.100.105*, the IP address that the smart-meter-api. The token *"f44f550a-3b08-482d-aa24-6a585286261e"*, is generated to identify the SM. On each subsequent request made by the SM, the token is provided to identify itself.

The host database saves the information in the Mongo database, as seen in Figure 3.11. The SmartMeterId contains a base-64 encoded version of the token. The Database presents the readings that is available for the MMS server to consume. The following section of this chapter provides the planning of the code and integration of the API with the SM. This section also consists of a partial implementation.

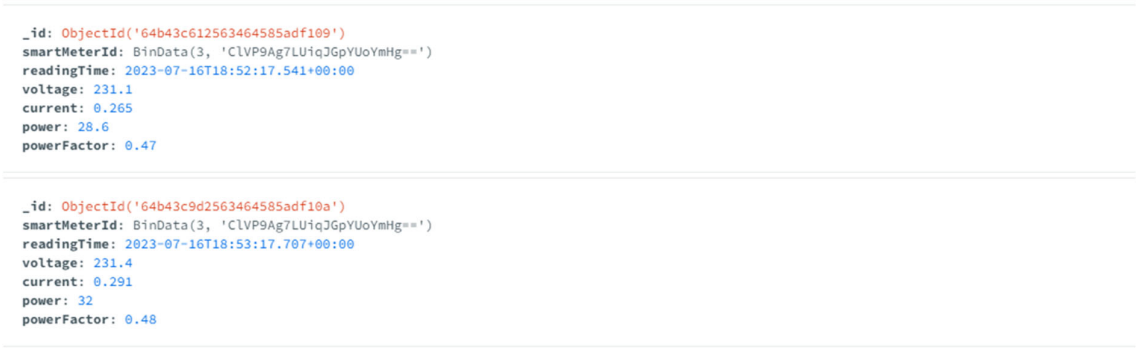

Figure 3.11: Database entries for the various readings that a single smart meter submitted.

3.3 Smart Meter API

3.3.1 SM API Overview

This section discusses the design and implementation of the API gateway and what is required from the API to accomplish the goals of the system. This section outlines the requirements of the other parts of the system, i.e., control centre and the SM.

The proposed system creates an intermediary for control centres/SCADA systems to receive information from the load with the SM playing the role of the residential load in this situation. All the SM readings are gathered and provided to the control centre in an organised, and standardised manner. Using an API allows the centralisation of data collection from all of the SMs. Using a polling technique these values are kept up to date as the meters are communicating with the API endpoints. The meter's transfer intervals are adjustable so as not to overwhelm the API with information.

All software applications are developed according to their business model. An application that requires many users to be active at a given time should be scalable. A

time-critical application should be developed with performance in mind. To determine what architecture to use, it is necessary to determine what the system needs from a performance standpoint. Using the items discussed in the literature review, it can be deduced that web service APIs are not only built for speed but also for interoperability between operating systems.

Looking at the power grid requirements, safety is one of the biggest issues when it comes to using electricity. The power grid can be dangerous, if emergencies are not handled in a timely fashion. The power grid must respond quickly to any mal-operation or fault. Time criticality is therefore a requirement for any architecture that is created with safety in mind.

The grid is constantly expanding to new geographical locations, so the API must be used to accommodate more and more SMs. In addition to being time critical, the API should also be scalable. More instances need to be created so that more SMs can use the API without sacrificing performance.

The clean architecture meets this project's requirements for an application that can uphold these requirements. The clean architecture is a software development pattern that emphasises separation of concerns, testability, maintainability, and flexibility.

At its core, clean architecture is about organizing code into independent layers so that changes to one layer do not affect the others. This is achieved by defining clear boundaries between layers and enforcing strict dependencies between them. In this way, the code becomes more modular, easier to test, and less coupled, making it easier to change and maintain over time.

Overall, clean architecture is a powerful approach to software development that emphasizes maintainability and flexibility. By following its principles, developers can create code that is easier to test, change, and extend over time, ultimately leading to more stable, scalable, and sustainable software systems.

The Architecture for the system being researched can be divide into 4 independent layers.

- Domain
- Infrastructure
- Application
- API

The domain layer is where all the business logic exists. In the case of the project, it contains the relationships between the control centre and the SM in a logical format. It contains database constructs, entity models of the SM and the control panel. The domain layer does not depend on technologies or frameworks as it is usually written in raw object-oriented notation.

The infrastructure layer is the support layer for the technologies required to build the system. This layer is a technical support for creating logic that integrates with external technologies to incorporate them into the application. Examples of this range from databases, caches, to integration with APIs from other applications. This layer is required to create caching and database integration to provide permanent and temporary storage for all data that passes through the system.

The application layer combines the business logic of the domain layer and implements the technologies of the infrastructure layer to create functionalities for the system. All other logic, aside from the domain/business logic, resides in the application layer.

The API layer is an interface that is used to connect to the system through the application layer. The API layer contains standards for connecting to the system via HTTP requests. Other interfaces can also be created to connect to the system. The following section outlines the various endpoints/integration points the API requires for a SM to submit its readings.

3.3.2 Endpoints

The API determines what application layer methods the external integrators have access to. APIs generally expose all functions created by the application layer to the outside world. This allows other systems to integrate with the API created. An endpoint is a single resource in an API. This resource can contain anything from executing a piece of code that creates a SM in the database to retrieving all the information that has ever been submitted to it. Each resource can be identified by a URL on the base DNS. The requirements for an endpoint can be categorized by its functionality. There are 4 main types of endpoints used in the SM API.

These endpoint types generally refer to CRUD (Create, Read, Update, Delete), and in HTTP language they are GPPD (Get, Post, Put, Delete). Each of these endpoint types has its own requirements that must be performed depending on the function. POST, for example, is used to insert data into the system and therefore may require a data body to be submitted with the request. The GET endpoint is used to retrieve data. A URL or header parameter may need to be specified to return a list filtered by the parameter. PUT is the update type endpoint, where the instance to be updated must

be identified. DELETE, as the name implies, deletes data, and requires an identification parameter to determine what to delete. This description of endpoint types is very general, and their use is solely based on application programming which is outlined in table 3.1. The application may have more use for these types than has been described here. See Figure 3.12 for a general description of the endpoints used in this study.

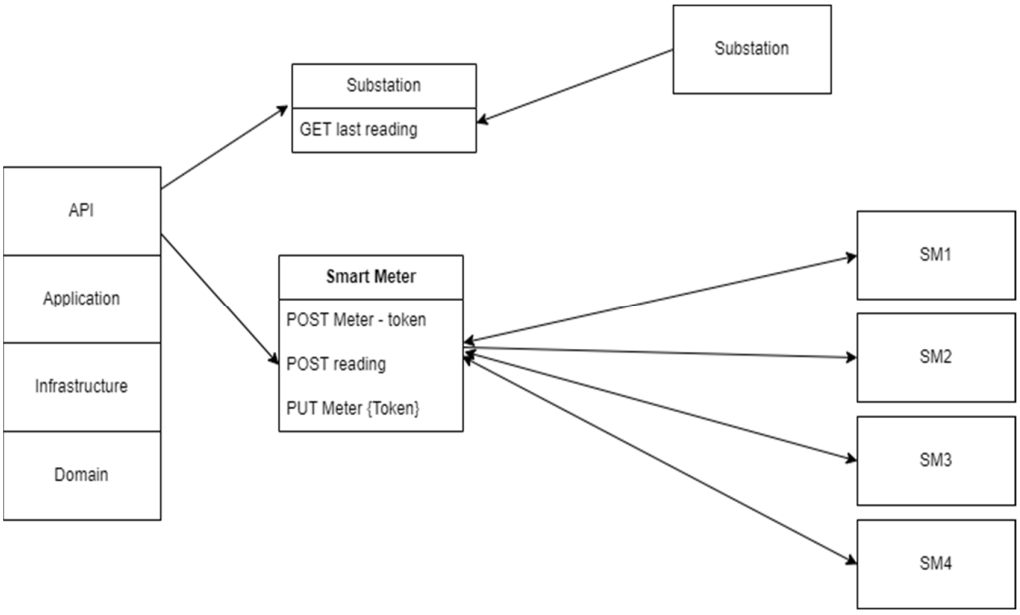

Figure 3.12: High level model of how the API Structure

Figure 3.12 shows the API Integration with the SMs and the substation/control centre/wide area management system. These endpoints are responsible for controlling the system and how it functions There are four endpoints related to the SM, each having their own distinct function.

1. *POST Meter* or */Meter*, this endpoint is the initialization endpoint. At the start of process, a meter needs to be initialised. *The POST <URL>/meter* endpoint is responsible for doing the initial handshake between the API and the SM.
2. Once the meter has received a token from the API, it is encrypted and stored into the database on the API to keep the SM data secure. The POST *<URL>/meter/{token}/reading* is the SM's way of uploading the data to the application.
3. The *PUT <URL>/meter/{token}* is used to extend commands to the SMs themselves. The meter regularly makes a call to the meter/{token}/status endpoint.
4. The *GET <URL>/meter/{token}/status* is used to allow a meter to get the configuration of the meter, to allow it to be controlled.

Figure 3.12 also shows the substation related endpoints; these endpoints are the interface which is used by a substation to integrate to the load.

- *GET <URL>/readings* provides the control centre with the latest single entry from each SM. This endpoint displays the different types of users of the data. The IEC 61850 standard supports XML as a data representation language. The JSON (Java Script Object Notation) is the norm in today's software engineering world. If the standard is to be updated to use the JSON instead of XML the endpoints already support the integration.

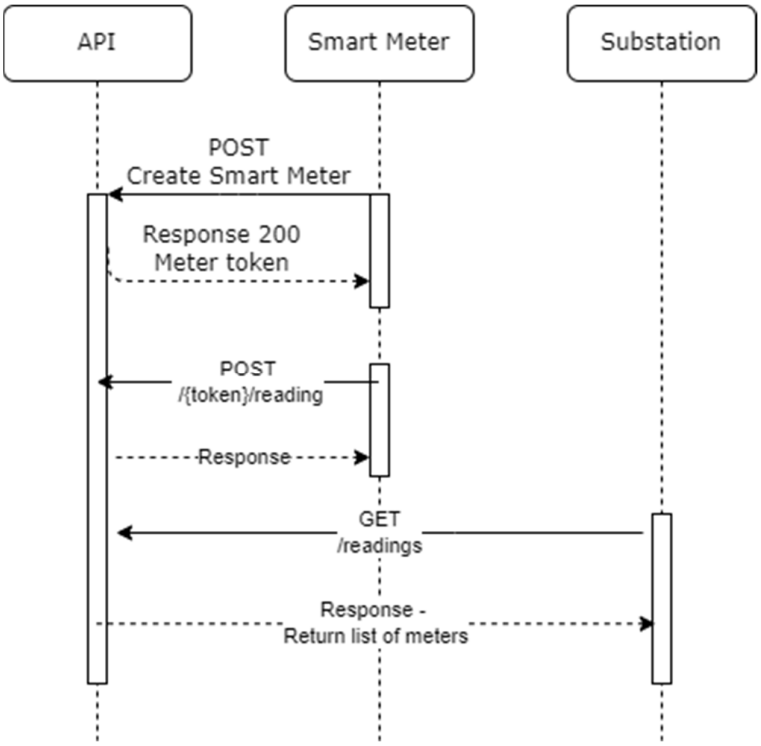

Figure 3.13: Sequence diagram presenting the main end-to-end flow of a SM

Figure 3.13 shows how these endpoints are used in a sequential process. The following steps are depicted in the above diagram.

- The process starts with creating a meter. The SM registers itself to the API. This action is depicted by the first arrow.
- The second arrow represents the HTTP response the meter receives. The SM analyses the response payload and retrieves a token from the payload. This token is the way of identifying itself to the system.
- The SM is then able to start reading power reading values. These values are submitted to the API by making a POST API call to "/{token}/reading" endpoint. As mentioned in section 3.2, the token is provided to the API to identify itself.

The API gives a “204 No content” Response to the SM indicating that it has successfully recorded its values into its database.

- The MMS server makes a GET “/readings” call. This allows it to retrieve the current values for all SM’s. The API then responds with all the readings contained in a JSON list format.

The rest of this section outlines how each endpoint works specifically with relation to response models, request models and the various types of data it is able to return.

The first endpoint called is the *“POST <URL>/meter”* endpoint. This endpoint is the initialization endpoint for the SM. This endpoint provides the meter and the system with a means of identifying the meter and classifying its readings. To create this token, the endpoint is supplied with information. To allow the end-user to classify the data into different categories. These parameters are given to the endpoint:

- Address
- Zip Code (Any form of address code not limited to American zip code)
- IP – Address

These parameters are specifically chosen to accommodate the end-user to make location-based decisions.

The second endpoint being used is the *“POST {token}/meter/reading”*, this endpoint contains a parameter named, token. This token is retrieved when a meter makes the *“POST /meter”* the API returns the token and stores it permanently in the API database to identify it in the future. This token is used to identify to whom which meter readings belong and log them accordingly. This POST request requires values to be submitted to it, to count as a successful reading.

- Voltage
- Current
- Power
- Power Factor
- Frequency

If not all of the above values are returned, the API attempts to convert the values into an IEC 61850 MMTR object, if not successful, it returns a message indicating that more values are required.

The third endpoint “*GET* /control/readings/” and “*GET* /control/reading/*{token}*” are the endpoints which main function is to return the metering data. These endpoints have

two different ways of returning data. The endpoint has two different sets of endpoints one for XML and one for the JSON. There are four endpoints in total related to the retrieval of readings. Making use of HTTP headers, the client can switch between the format of the response message by changing the header to the expected format.

The ‘accept’ header determines if XML or JSON is being returned to the requesting application. By default JSON type formatting is provided for all readings. Otherwise, an XML file is returned with meter readings in a .SCD format.

- GET /control/readings/xml
- GET /control/readings/json
- GET /control/reading/{token}/xml
- GET /control/reading/{token}/json

The GET endpoint with the token only retrieves the latest reading and the token determines which SM’s values is being requested. The values being returned depends on the sensor. The ideal sensor returns:

- Voltage
- Current
- Power
- Power Factor
- Frequency

In case the sensor does not return all the above values, the API stores the values that are returned. For example, if the sensor is not designed to return frequency or power factor the API does not save those values and makes the values *null*.

Table 3.1: List of endpoints

REQUEST TYPE	END-USER	URL	Request Body	Response Model
GET	Control Centre	/control/readings/xml	n/a	JSON{LIST[Voltage, Current,Power,PF, Frequency]}
GET	Control Centre	/control/readings/json	n/a	LIST[MMTR]
GET	Control Centre	/control/reading/{token}/json	n/a	MMTR
GET	Control Centre	/control/reading/{token}/xml	n/a	JSON{Voltage,Current, Power,PF,Frequency}
POST	SM	/meter/	Country, City, Street Name, Address Code	{token}

POST	SM	/meter/reading/{token}	JSON{Voltage, Current,Power, PF,Frequency}	N/A
DELETE	SM	/meter/{token}	n/a	N/A

Table 3.1 shows a summarized table of the list of endpoints required to enable system integration to the various parts of the system. The table uses a notation where LIST [], JSON {} and MMTR is used. LIST [] is used when a LIST of values is returned, for example LIST [MMTR] returns a list meter reading. JSON {} indicates that the response or request is which is returned in XML format being returned in a JSON format. MMTR is generally the logical node following the IEC 61850 guidelines for transferring information.

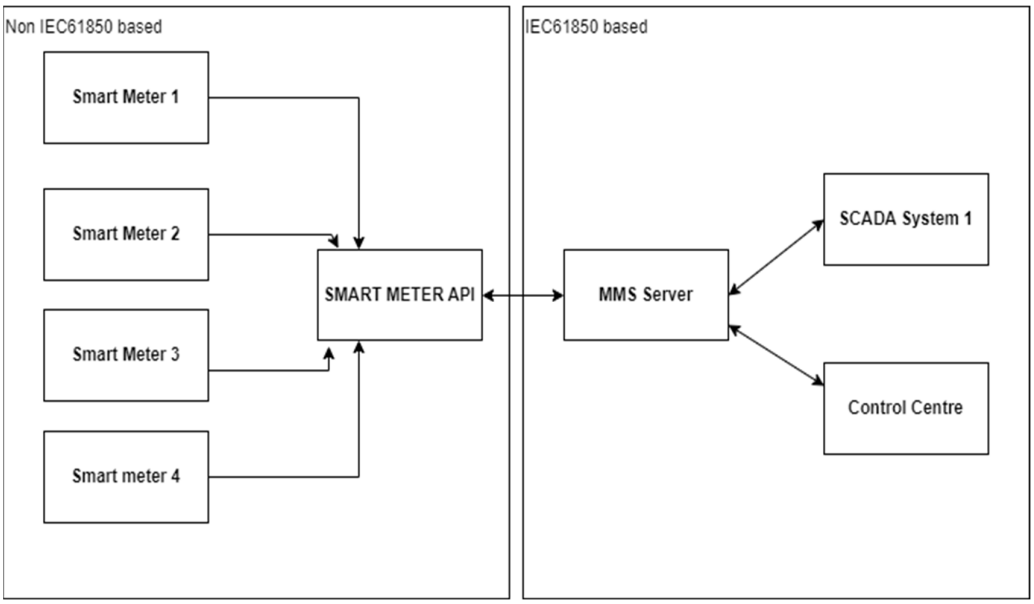

Figure 3.14: Representation of the API and SM system in an IEC 61850 environment

Figure 3.14 shows the relationship between the substation, control centre and the system that is created. From a substation perspective the system should look like another substation having metering as its main function.

The “smart-meter substation” is designed to have an API Gateway and MMS interface as the interface to the entire “smart-meter substation”. The system is designed to act like a substation with a gateway device. In any IEC 61850 environment the MMS server ensures that the “smart-meter substation” looks like an actual substation, with various metering devices. The following section presents the partial implementation of the Smart Meter API, to create a sandbox to test the integration between the various components.

3.3.3 Implementation of SM API

The creation of APIs has been simplified in today's world. Each OOP (Object Oriented Programming) language has some sort of library or framework that can accommodate APIs. The challenge with APIs is that they must be maintainable and scalable, while being secure at the same time.

This research work does not require a vast number of APIs to be deployed to allow the system function. The API's use is singular and does not target anything other than SMs and control centres. The goal of the API is to get data, format data and provide data to the necessary project.

The formatting of the data requires the API to format data according to the directives that the IEC 61850 standard outlines. The API created, is in a RESTful standard to allow various applications to integrate with the API. Thus, IEC 61850 compliant devices that receive data submitted via REST API require formatting to XML for conversion to the IEC 61850 standard.

An SQL database is utilized to create structured data, that can be manipulated or retrieved based on a query provided. In a real-world scenario, the use of a NO-SQL database such as MongoDB would not be ideal since the data would need to be structured correctly for easier consumption using SQL commands.

The project is created in this GitHub repository: https://github.com/RyanKruger1/smart-meter-api.git. The above repository implements the SM API and the various clients that are required. The API has been written in .NET, a very common middleware application programming language. The clients are written in the Python programming language due to its various interoperability across multiple operating systems, including the Raspberry operating system – the main operating system for the Raspberry Pi.

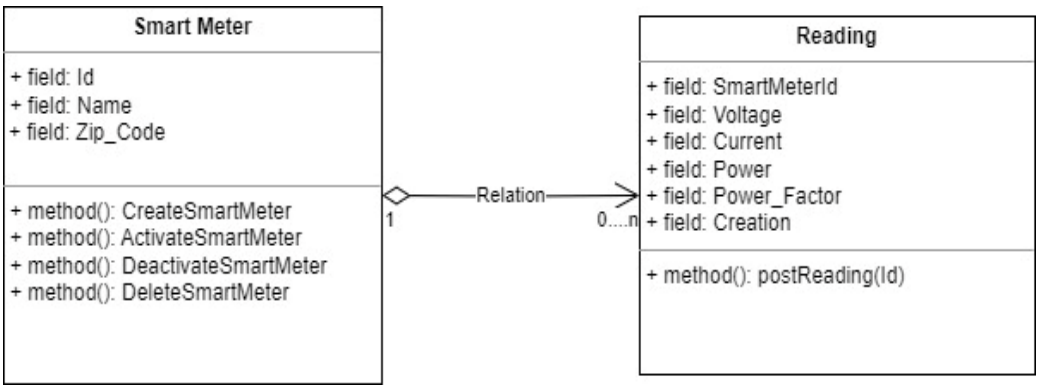

Figure 3.15: Entity Relation Diagram of the SM and readings Classes

Figure 3.15 shows the main model relationships being utilized by the SM API. These models are the basis of how the data is stored in the NO-SQL database. The SM has a one-to-many relationship with *reading*. Reading has an identifying field named “creation”. The “creation” field is the main identifier of the readings, it is a date-time combination, which are forever unique. Each reading is associated to a SM using the “SmartMeterId”.

API endpoints are created to accommodate both a SM integration and a control centre integration. Figure 3.15 depicts the relationship that is used to create endpoints that can accommodate any relevant parties. The above models are mainly used to create the basis of the relationship between the two entities. As previously mentioned, the API software is split into four layers, i.e. API, application, domain and infrastructure. These models live on the bottom layer of the software architecture, i.e., the domain layer. This allows the entire system to be built upon these models.

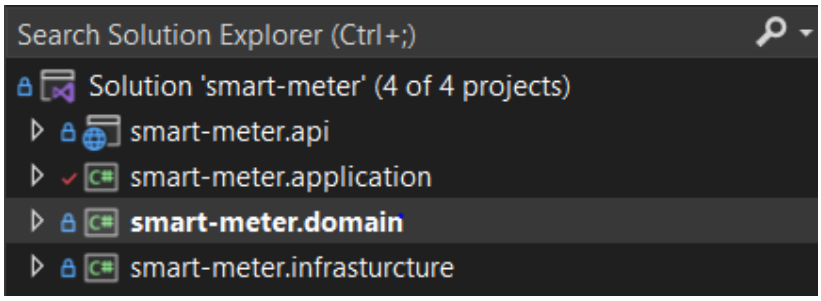

Figure 3.16: The 4 layers of the smart meter-API

The domain layer contains the models, and any other models that are required to create the API. The infrastructure layer contains any infrastructure related dependency which requires outside integration. The SQL database connection of the API resides in the infrastructure layer. Any additional service that requires an outside connection resides in this layer.

As the name suggests, the application layer is where these models are applied to create methods to allow the fulfilment of the goals of the API. The API layer is where the application layer to the public using HTTP methods are exposed.

The application layer contains various methods to implement all the required functionality. Since all the information is already stored in a structured manner, the purpose of the application layer is to:

1. Expose it to the public layer
2. Format all the data to conform to the IEC 61850 standard.

Using Docker, a containerization engine, any port of the local machine is deployed to act as the server interface. ASP.NET provides a package for testing API endpoints called Swagger. Swagger is enabled by the SM API in development environments to allow for easier testing of the endpoints. The following URL allows access to the Swagger UI to view the available endpoints:

http://localhost:<PORT>/swagger/index.html

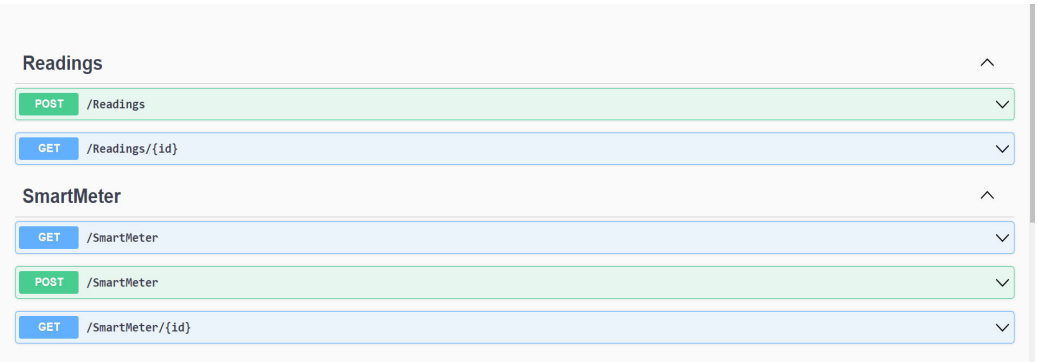

Figure 3.17: Page displayed from swagger to document API

The page delivered is shown in Figure 3.17 - notice that it splits up the endpoint according to where their controller class lives. The figure below shows the outline of the exposed smart-meter API.

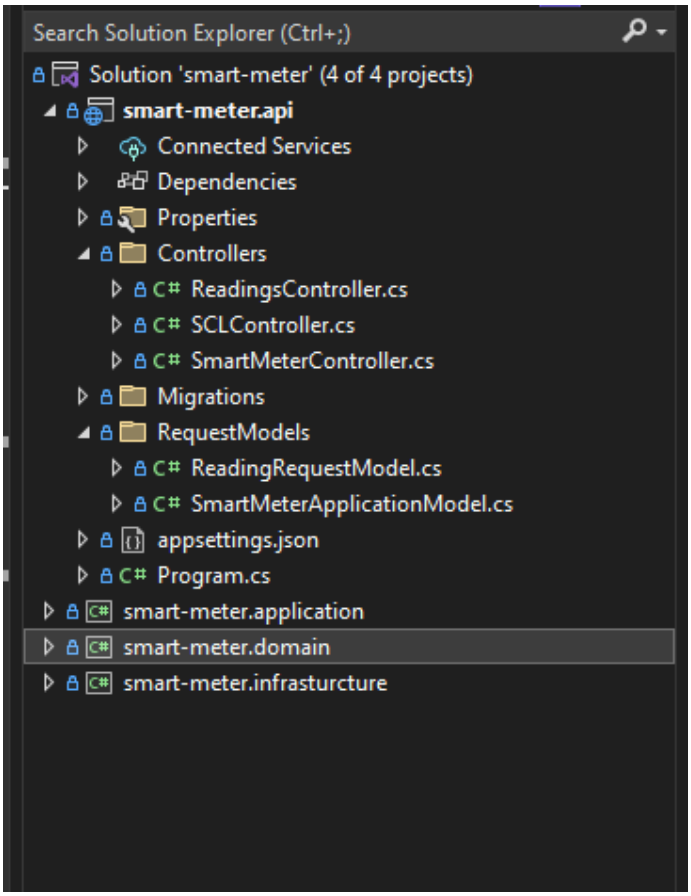

Figure 3.18: Folder structure of the API project

Each one of the endpoints above is configured to have an *Accept* header; either the JSON or an accept XML header. The response returns the format that is selected via the header being passed into the call; by default, the request and response set to respond with JSON. Once the GET call is made with the accept header being set to “XML”, the response will no longer be JSON, instead it is in the XML format.

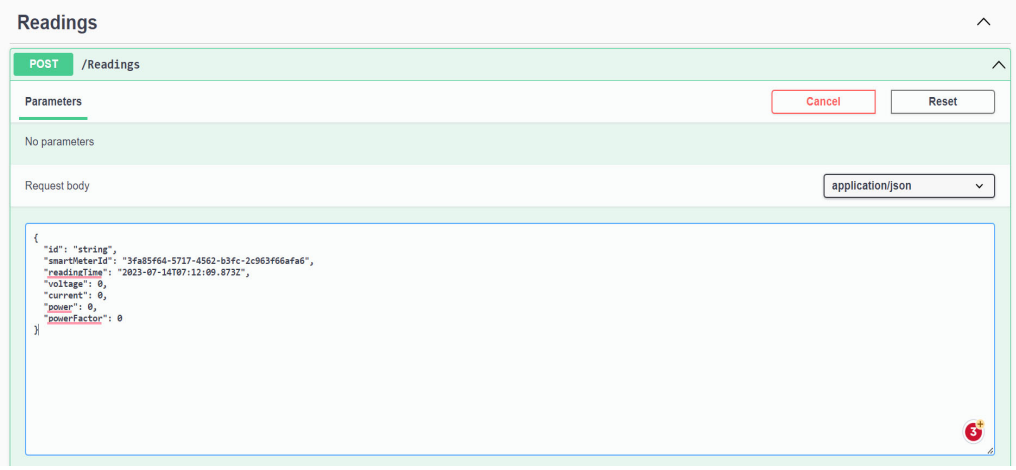

Figure 3.19: Request requirements viewed by Swagger

Figure 3.19 shows the endpoint that is most widely used in this research work. As mentioned earlier, the token or “smartMeterId” is provided to the endpoint via the JSON body shown above, along with the various metering values such as voltage, current, etc. The following section introduces the planning of the MMS server that is used for integration in this research work.

3.4 MMS Server

The IEC 61850 standard provides a great amount of input regarding the future proof design of communication and control in the Smart Grid (SG). The focus of the IEC 61850 standard is to achieve interoperability between devices from different manufacturers in the SG. The standard allows for seamless communication between the parts of the grid, mainly generation, transmission, and distribution. The load has not been looked at in such generalised detail, as it is impossible to generalise it due to its variance.

The proposed system in this research work is to demonstrate the integration of the load into an IEC 61850 environment. The group of Smart Meters (SM) that are integrated to act as the load is structured to look like a substation with an MMS (Manufacturing Message Specification) server. The control centre or any grid-management can connect to the load/substation via MMS, without ever knowing that the components are not part of any substation, but a virtual substation. The IEC 61850 standard does not document the use of advanced metering infrastructures; therefore, an AMI (Advanced Metering Infrastructure) is modelled after a substation.

Figure 3.14 shows the different components of the virtual substation and relationship within the system. The MMS server is the IEC 61850 based interface to the virtual substation, while the Smart Meter API is the generic API interface.

Each SM is virtually identical to all other SMs in this system, because they all are created from the same template. The only aspect that separates the SM, is the token controlled by the Smart Meter API that identifies each SM. Each SM is designed to have 3 logical nodes.

- CSWI – Control for XSWI
- MMTR – The metering logical node
- XSWI – The control for the circuit breaker on which the SM is located.

The logical nodes are not implemented on the SM itself. The SM reports its values to the Smart Meter API. The API provides an XML-based Substation Configuration Language Substation Configuration Language (SCL) file as defined in IEC61850-6 on the current state of the virtual substation to the MMS server. The MMS server exposes all information to any control centre/SCADA system connected to the server.

3.4.1 Data Construction

The virtual substation has the main purpose of supplying any connected client with IEC 61850 objects. The MMS server provides this connection to external parties.

The IEC 61850 standard has various communication protocols. MMS is one of the few protocols that stretches beyond a substation. The MMS standard provides the SCADA system with an interface into substations, in addition to external components.

This project utilizes a Java project created by beanIT (2020). This project provides a client-server application for integrating with or hosting an MMS server. The virtual substation that is created, hosted on an MMS server to provide this integration with external parties. The only requirement is the construction of the SCL files. Since the smart-meter-api is connecting SMs dynamically, it is required to continuously updated the SCL file in the Java library memory.

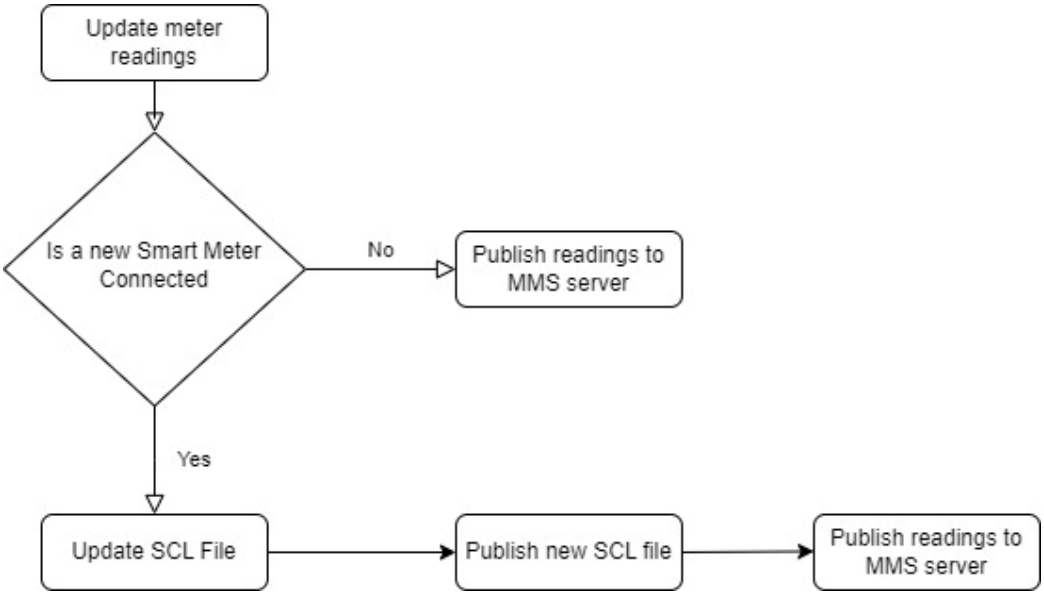

Figure 3.20: High level logic overview of API-MMS server

Figure 3.20 displays the logical flow of the dynamic SCL file. This is to accommodate a new SM joining the neighbourhood with the system being able to accommodate this addition. As soon as a change is detected a new SCL file is generated to keep the MMS server running on the virtual substation up to date. The above diagram shows the

system cycle that takes places. Every cycle the system attempts to update the meter readings. The updating of meter readings has a time-based update, for example every 500ms. This value is configurable based on the needs of the MMS server.

Once a cycle starts, the system attempts to populate the MMS server with the latest data. It also checks if new meters have been registered with the API. If a new meter has been registered a new SCL file is published, and the MMS server loads that new meter with the readings that is communicated by the meter.

3.5 Conclusion

In conclusion, the construction and deployment of SMs, with a well-planned API and seamless integration of the IEC 61850 MMS server, represent a significant step forward in modernizing our energy infrastructure. These advanced meters not only provide more accurate and real-time data on energy consumption but also enable utilities to enhance their grid management, reduce operational costs, and empower consumers with greater control over their energy usage.

This section demonstrated and tested the various parts of the system individually. The SMs components are tested to make sure that the information is retrievable and usable by the rest of the system. It also helped in planning what would be the centralized processor for the SM. The data the power reading sensor reads is used to plan the API’s request and response models accordingly. The MMS library that is used to facilitate the IEC 61850 interface is presented and the integration between it and the API is discussed, modelled and planned.

The following chapter makes use of the above planning, documenting reports on the implementation and testing phases of the SDLC. This chapter is used as a guideline to solve unforeseen problems and creates solution that will achieve the proposed system goals.

CHAPTER 4

IMPLEMENTATION AND INTEGRATION OF THE SMART METER SYSTEM

4.1 Introduction

This chapter presents the previously discussed topics and methodologies to implement a system that can achieve the goals discussed in the Introduction chapter.

This chapter follows the implementation stage of the SDLC (Software Development Life Cycle). This chapter uses the planning phase presented in the previous chapter to integrate each system component. This chapter presents the integration steps that is required to achieve the goals set out in Chapter 1.

In the earlier chapter a switching mechanism is planned, to create the power switch feature. After the system was implemented, it was realised that the switching functionality would require work, not only from an IEC 61850 perspective. It is decided to rather focus on the metering aspect of the Smart Meters (SM).

The implementation chapter is split into five sub-sections. Each sub section implements a different part of the system. Section 4.2 discusses the environment, which describes what the computer architecture looks like for the entire system and the platforms on which the system is run. This section is mainly to describe the hardware being utilized in the system. Section 4.3 presents the Message Manufacturing Specification (MMS) server configuration and setup. This section additionally consists of the discussion of the REST API, MMS Server and the caching mechanism. Section 4.4 describes the S's software and integration with open-source components. Section 4.5 details the SM client, meter module and SM simulator. Section 4.6 provides the conclusion to this chapter.

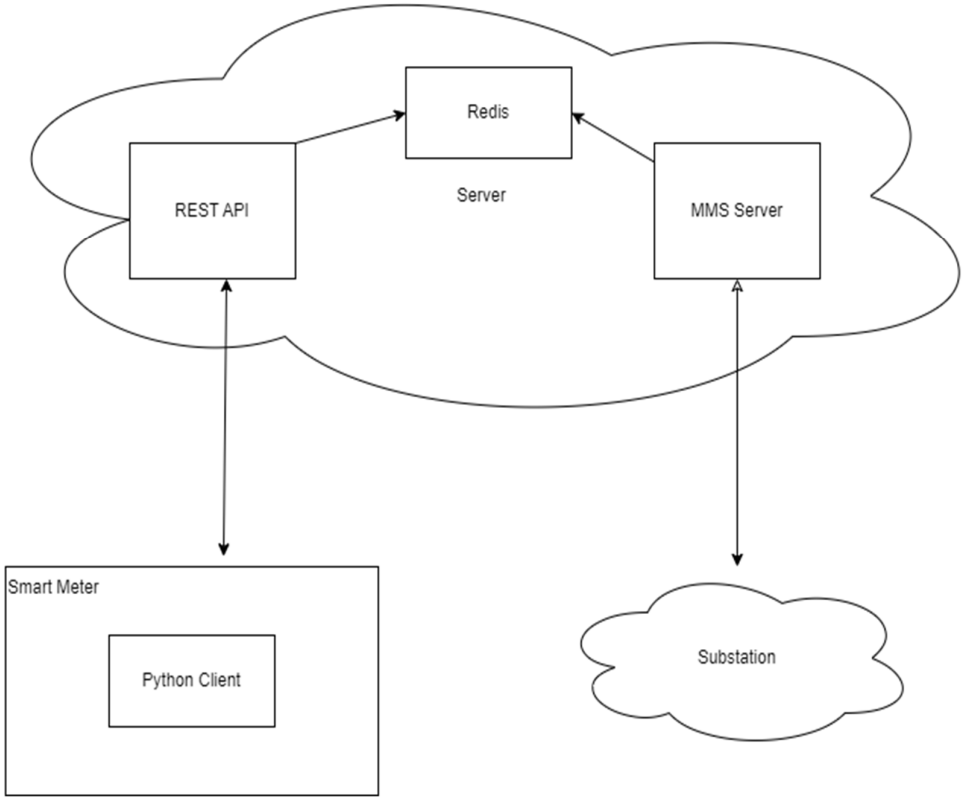

Figure 4.1: High level overview of system

Figure 4.1 shows the layout of each part of the system. The SM (containing the Python-based client) communicates with the Redis server through the REST API, while the MMS server communicates to the Redis server and substation devices. The implementation of the environment is a pivotal step in ensuring that the entire system operates cohesively and efficiently. As this chapter, progresses, the technical intricacies, best practices, and considerations that went into setting up an environment conducive to achieving the research goals are explored. By establishing a robust and well-designed environment, the stage is set for gathering data, analysing system performance, and ultimately determining the success of the research work. The environment includes a couple of aspects, like the networks which is used to achieve communication amongst the components of the system, the hardware, on which all the software components run, and the interface and operating systems the hardware uses.

4.2.1 Network Setup

Each part of the system is in a different part of the communication network. The server software shares a system to allow efficient communication between the various software platforms.

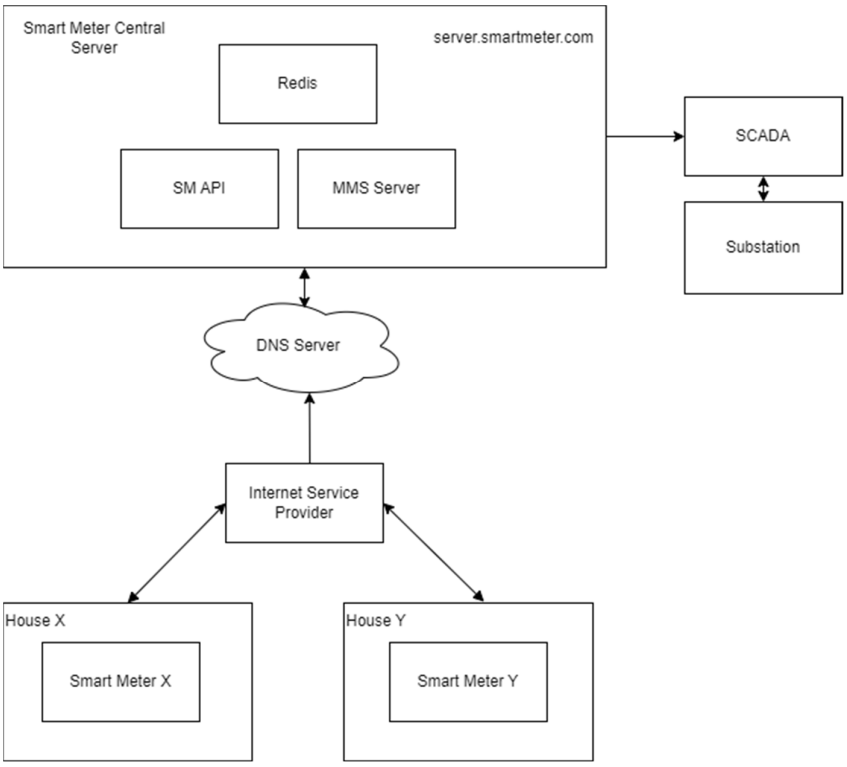

Figure 4.2: Network Layout

Figure 4.2 shows the various communication networks involved in the system. The SMs live on their own local area networks, which is seen having their own local IP addresses. The web server itself is hosted online and is identifiable with its domain name.

The web server is a lightweight version of a normal server. A normal server is a system that contains components for distinct roles. For example, RAM (Random Access Memory) modules for memory, hard drives for storage dedicated Graphical Processing Unit (GPU) and CPU for processing.

4.2.1.1 Web Server hardware

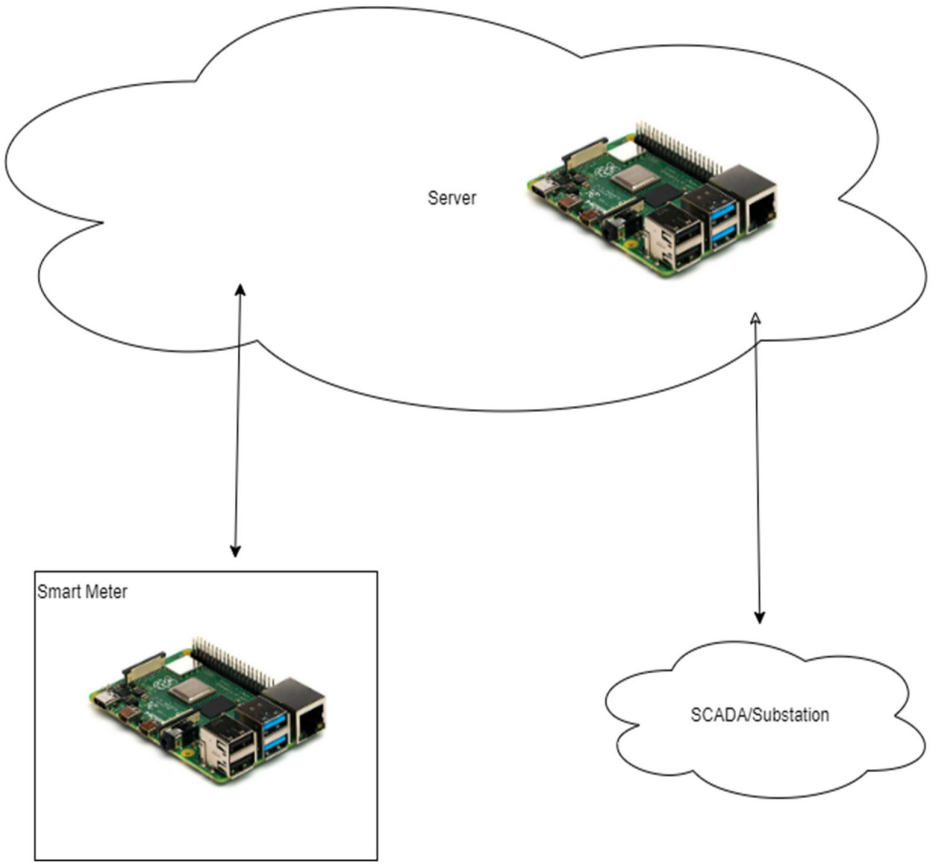

Figure 4.3: Hardware representation of system

Figure 4.3 shows a hardware representation of the system. It is seen that both the server and SMs are being facilitated on Raspberry Pi development boards. This figure demonstrates that the server components all consist of Raspberry Pi boards. The only thing that distinguishes the two types of components is the type of software that is being run.

The server from Figure 4.3 represents the Smart Meter API and the MMS server. The SM is also represented by a Raspberry Pi development board. The SM to Raspberry Pi ratio is 1:1. Each SM consists of its own Raspberry Pi board, with a power reading module attached.

Figure 4.4: Raspberry PI Server Rack

Figure 4.4 shows the lightweight server used to accommodate the MMS-Server and the Smart Meter API. The server has a Raspberry Pi – marked by A, an external hard drive marked by B and a network switch marked by C.

Figure 4.5 outlines the logical connections of the server, with respect to the Internet. The SMs communicates with the server via the Internet. The processor and the storage controller's duties are both being fulfilled by a dedicated Raspberry Pi board.

The following section discusses the software environment that uses the previously described hardware and communication networks.

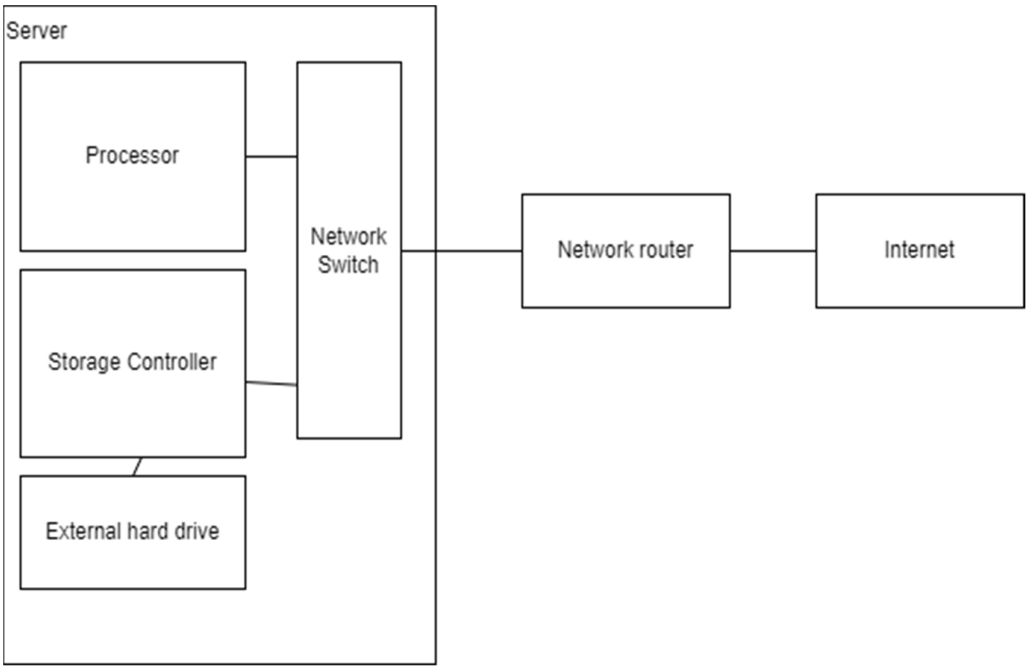

Figure 4.5: Logical network connection

4.2.2 Device Setup

The Raspberry Pi operating system is created to facilitate a UNIX-based operating system for the credit card sized computer. All Raspberry Pi development boards being used in this research work are equipped with this operating system.

The Raspberry Pi organization hosts imager software on the company's main website. This imager is used to upload Raspberry operating systems onto any form of storage devices. All components in this research work make use of SD cards, not only as storage, but as the operating system container.

4.2.2.1 Installation of Raspberry Pi OS Lite (64-bit)

To set up the Raspberry Pi, the Raspberry Pi OS Lite (64-bit) is installed a on an SD card. This is achieved by downloading a software imager from the Raspberry Pi website (Pi, 2023).

The imager requires a connection via either USB adaptor or a memory card adaptor. After inputting one of the above options into the computer, the imager software should be run and the operating system "Raspberry Pi OS Lite (64-bit)" should be selected along with the driver on which the OS (Operating System) is downloaded to.

The Raspberry OS lite is a lightweight version of the normal Raspberry OS. The lite version of the operating system only contains a Command Line Interface (CLI), compared to the graphical user interface the regular operating system contains. This also results in better performance due to the minimalistic use of the operating system.

The intended use of this project is to use the Raspberry Pi as servers, operating them either remotely or by exposing the ports of the computer. The Lite version of the operating system is perfect because it allows for remote connections without the use of an operating system.

4.2.2.2 SSH

As mentioned above the Raspberry Pi OS lite operating system does not have a graphical user interface. The Raspberry Pi development boards used in this project are configured and communicated with by making use of a built-in interface. The Secure Shell (SSH) protocol is a remote command line client into a Unix based system. This form of communication makes use of crypto graphic algorithms to secure communication. It can also be used to communicate with devices on the same network remotely.

4.2.2.3 Docker Containers

All the projects that are created in this research work is containerized with Docker, to allow full transferability and easy deploy ability.

Docker is a prominent containerization technology that has changed the way applications are developed, deployed, and managed in modern computing environments. It provides a comprehensive platform for creating, distributing, and running applications within containers. A container is a lightweight, standalone, and executable package that contains everything needed to run a piece of software, including the code, runtime, system tools, libraries, and configurations. Docker allows these containers to be created, moved, and executed consistently across different computing environments, offering several key advantages:

Docker containers encapsulate an application and its dependencies, ensuring consistency across various environments. This portability allows developers to build and test applications on the local development machines and then deploy those containers to production servers or cloud platforms without worrying about compatibility issues. This standardization simplifies the development and deployment process.

Docker containers provide a level of isolation between applications and their host system. Each container operates independently of others, which prevents conflicts and ensures that one container's activities do not impact the stability or security of other containers or the host system. This isolation is achieved through containerization technologies, such as namespaces and groups, which are part of the underlying Linux kernel.

Docker facilitates the dynamic scaling of applications. Containers are easily replicated and deployed to meet varying workload demands. This scalability is especially useful in cloud-native and microservices architectures, where applications can automatically adapt to fluctuations in traffic or resource requirements.

Compared to traditional virtualization, where each virtual machine (VM) requires its own full operating system, Docker containers are much more efficient. They share the host operating system's kernel, leading to reduced overhead and lower resource consumption. As a result, more containers are hosted on the same hardware, optimizing resource utilization.

The Docker commands and Docker compose files are presented which encapsulates all the applications that are required to allow the various application to run independently and to connect when configured to do so.

Docker is responsible for the management, deployment and connection of the various applications that are being used. The various concepts related to Docker are explained in the Table 4.1 below:

Table 4.1: Docker concepts (docs, 2023)

Docker Term	Description	Example Usage
Docker Image	A Docker image is a lightweight, standalone, and executable package that contains an application and all its dependencies, including code, runtime, libraries, and configurations. Docker images serve as templates for creating Docker containers. They are typically built from a Docker file, which defines the image's content and settings.	Creating an image for a web server that includes the web server software, website files, and configuration settings.
Docker Container	A Docker container is an instance of a Docker image. It is a runnable environment that isolates applications and their dependencies. Containers operate independently and share the host system's kernel. They are portable and can run consistently across different environments, making them suitable for microservices and application deployment.	Running multiple containers from the same web server image to host different websites, each in its isolated environment.
Docker Daemon	The Docker Daemon, or Dockerd, is a background service responsible for managing Docker containers. It listens for Docker API requests and handles container operations, including building, running, and managing containers. The Docker Daemon communicates with the Docker client and interacts with the underlying system's resources.	Automatically starting and managing containers based on images when a request is made to a web application hosted by Docker.
Docker Engine	Docker Engine is a comprehensive solution that includes both the Docker Daemon (Dockerd) and the Docker client (Docker CLI). It is responsible for container management, orchestration, and communication with container registries. The Docker Engine is a core component of Docker's architecture, providing tools for developing and deploying containers.	Using Docker Engine to create a cluster of containerized micro-services that are easily scaled up or down to meet varying demands.
Docker Compose	Docker Compose is a tool for defining and running multi-container Docker applications using a simple YAML file. It allows you to specify services, networks, and volumes for an application, making it easier to manage complex applications consisting of multiple containers. Docker Compose simplifies the deployment of interconnected services and applications, particularly for development and testing environments.	Defining a Docker Compose file to set up a development environment with multiple containers, including a web server, database, and caching server, all working together.

Making use of Docker images all software requirements, like database such as Azure-edge and PostgreSQL databases are loaded onto the system without having to download software. Instead, a Docker image is *pulled* that already contains these software's.

Once a Docker image is run, the Docker Daemon creates a container based on that image. This container is assigned a port on the target computer which is used to interface with the relevant software. The command snippet below, shows a bash terminal command that is run to start a version of MySQL. It is exposed to port 3306 and all information is stored on that system with the path directory as set to "/path/on/host/".

bash Copy code

```
docker run -d \
  --name mysql-container \
  -e MYSQL_ROOT_PASSWORD=my-secret-pw \
  -p 3306:3306 \
  -v /path/on/host:/var/lib/mysql \
  mysql:latest
```

Figure 4.6: Docker compose file for MySQL database

Making use of Docker-compose.yml files all the applications (MMS Server, Smart Meter API and Python software controller) are deployed and run using Docker compose file.

Running *"docker container ls"* shows the various containers that are currently running. Figure 4.7 shows all the applications that are currently running on the machine via Docker.

ryan@raspberrypi: ~

```
ryan@raspberrypi:~ $ docker container ls
CONTAINER ID   IMAGE                              COMMAND                  CREATED      STATUS
1d501cacb667   sm-dashboard-angular-app           "docker-entrypoint.s…"   8 hours ago  Up 32 minutes
a4bbddcc62e1   s-sm1                              "dotnet smart-meter.…"   9 hours ago  Up 33 minutes
1a83c20d1a5c   mcr.microsoft.com/azure-sql-edge   "/opt/mssql/bin/perm…"   9 hours ago  Up 32 minutes
5ea4271605f4   portainer/portainer-ce             "/portainer"             9 hours ago  Up 33 minutes
```

Figure 4.7: Docker container ls – results

It is seen from Figure 4.7 that the executed command shows the image on which each of the containers are based upon. It also gives valuable information regarding which

ports of the virtual machine is mapped to the actual ports of the Personal Computer (PC).

There are two main roles in this implementation section. The first one being the server which is depicted in Figure 4.8 with 4 layers. Looking at the diagram bottom-up, the infrastructure layer represents the connection to the network and the connection with other resources on the network. The operating system as previously described uses the Docker layer as a base. The Docker layer stores the images of the REST API and the MMS server. When given instructions the Docker Daemon constructs containers of the REST API and MMS Server upon itself. Exposing ports to the infrastructure layer for other devices to consume.

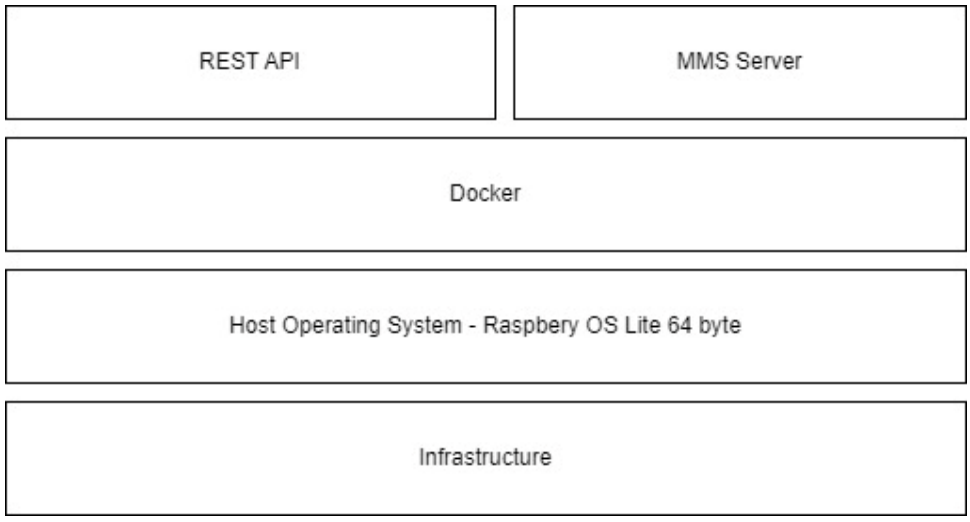

Figure 4.8: Server technology stack

The SM technology stack, depicted in Figure 4.9 is almost identical to that of the server, except for the containerized layer which in this case contains the Python script responsible for communicating with the infrastructure layer and retrieving information from the sensors.

In Figure 4.9 all layers have different projects associated to them. There are 3 main projects each having their tools to start the application running and keep it running as a server. A Docker file is created for each one of the above applications to allow them to be run from a Docker image. A Docker file is the file that outlines how a Docker image is constructed. Subsequent application requiring the software pulls from this Docker image defined by the Docker file. Once the Docker file is created the command: *docker build* is run.

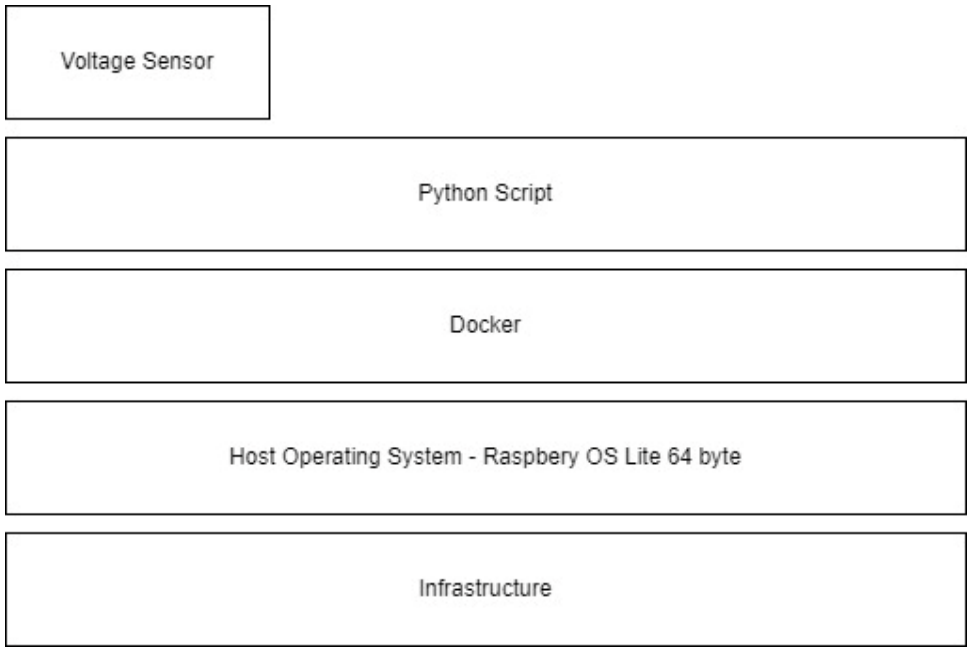

Figure 4.9: Smart Meter technology stack

Dockerfile Copy code

```
# Use an official Node.js runtime as a base image
FROM node:14

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install app dependencies
RUN npm install

# Copy the application code into the container
COPY . .

# Expose the port that the app will run on
EXPOSE 3000

# Define the command to run your application
CMD ["npm", "start"]
```

Figure 4.10: Code snippet of a Docker file

Figure 4.10 shows an example of a Docker file, all files have different steps that creates the application. All server type application has a final line like that of figure 4.10. The CMD command starts a command line and executes the commands with parameters.

For example, above the command that is executed is ***"npm start".*** Below is an outline of the commands that the system uses to allow all applications to be converted to Docker files.

Table 4.2: Command line commands to execute programs

Application	Command	Programming Language
MMS Server	java -jar mms-server.jar	Java
Smart Meter API	dotnet publish.	C# - .Net
Smart Meter Client	python3 smart-meter.py	Python

The following section introduces the MMS Server implementation, where each component contains its own section to discuss its implementation and various other requirements in order to achieve the best possible implementation.

4.3 MMS Server

4.3.1 MMS Server introduction

The MMS server that resides in a server/client MMS Java repository, is introduced by a company named BeanIt. https://github.com/beanit/iec61850bean , is a Java repository the library comes with a pre-built asn.1 script. This script creates Java class to accommodate the MMS technology stack. Figure 4.11 shows an ASN.1 (Abstract Syntax Notation.1) folder which contains the script to generate the correct presentation layer stack that is outlined in the IEC 61850 standard for the implementation of MMS.

Making use of the Intellij IDE, the repository is run using its package manager software, Gradle. Gradle is responsible for running the various tasks regarding the code base. It is also responsible for downloading all the dependencies. It allows basic programming of task to allow integration into DevOps systems or the creation of .jar files. The .jar files are used to run Java applications.

Figure 4.11 shows the project hierarchy for IEC61850bean in the Intellij IDE. It is seen that there are 3 classes which contain main activities. These classes are *ClientGui*, *ConsoleServer* and *ConsoleClient*.

These main activities are the entry points to run the various features the repository provides. The server and the two client programs contain the main activities. These main activity Java classes are injected into the .jar files to allow the running of a specific main activity.

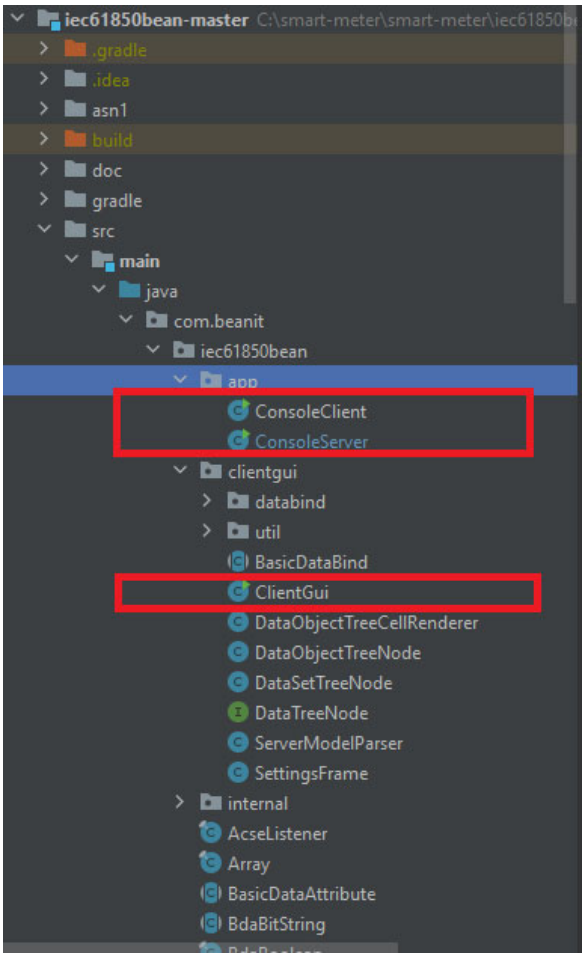

Figure 4.11: IEC61850bean project hierarchy

To create a jar file of the project the package manager software, Gradle needs to be run with the correct set of arguments. The creators of the project created a task that allows for the generation of .jar files. *Gradlew –jar <mainfile.java>* creates a .jar file which main activity is outlined by the mainfile.java

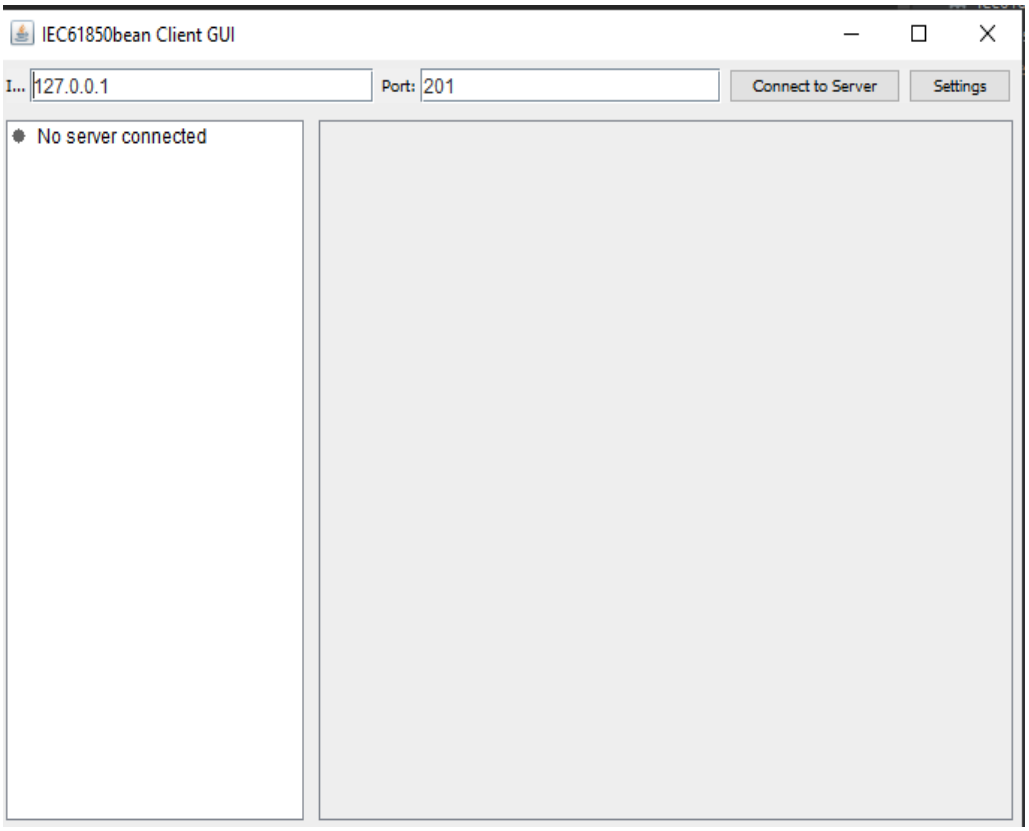

Figure 4.12: MMS server connection user interface

Figure 4.12 shows the graphical user interface developed to allow a user to connect to any MMS server on that particular network. Providing the IP Address and the port allows the interface to connect to that server. Clicking connect to server makes handshake and connects to the server.

To run the server the server requires two parameters, the model file and the port. To start the Java server the -jar iecbeanfile -p 201 -m /src/test/resource file is required.

The following section discusses the software, that is implemented to solve a performance issue.

4.3.2 Redis

Making use of *IEC61850bean's* MMS server implementation creates limitations due to the goal of the implementation not including dynamicity. The MMS repository is created to read the Substation Configuration Language (SCL) file only once. Within the developed system, once a new SM is added it is required to reload the MMS server to allow the new SM's to also be interrogated.

The SM REST API can generate a SCL file that is capable of being read by the MMS server. The problem with generating an SCL file, is that it consumes space on the host system. The bigger the file becomes the more space it consumes. This is where the

introduction of a Redis server comes into play. Redis is used to temporarily contain the SCL file contents until it has been read and the content is removed from the system.

Redis is a caching technology that is commonly used in REST APIs to increase the performance of an API. Redis is mainly used to allow too fast retrieval of records that doesn't change much. This increases the performance of API's because a database technology's RDBMS (Relational Database Management System) doesn't need to be interrogated, to retrieve data that is always being retrieved.

Redis uses a key value pair system that assigns a key to a value. The Servers uses this key value pair system to communicate with SCL files. Making use of the key "SCL_CURRENT", the server either uploads or retrieves this value.

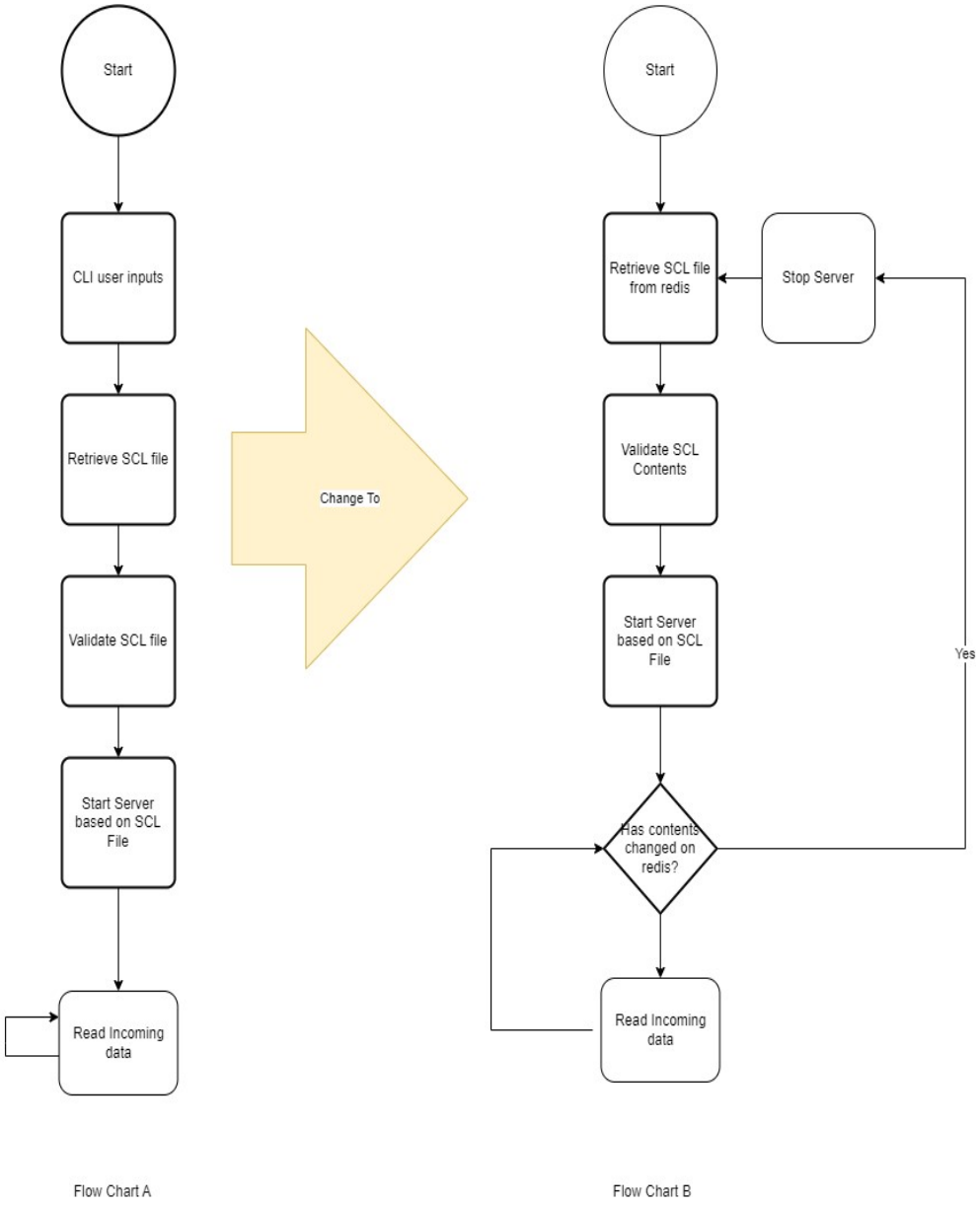

Figure 4.13: Flow chart of change Redis changes to system

The implemented system, however, should be able to dynamically update its SCL file and the contents of the MMS server to allow the addition of new SMs entering the system. Figure 4.13 outlines the flow charts of each of the implementations. Flow Chart A shows *IEC61850bean's* original implementation of the MMS server. Flow Chart B shows changes made to *IEC61850bean's* repository to allow the dynamic addition of SMs to the server.

[FLOW CHART A]: *IEC61850bean's* original implementation of the MMS server software.

[1] The Redis server is interrogated to find the current SCL value.

[2] The SCL file is retrieved and read into the memory of the application.

[3] The SCL file is validated to ensure that it conforms to the IEC 61850 standard.

[4] The Server is started.

[5] The server reads inputs provided

From the flow chart description above it is seen that after the server is started there is not attempt to check if new devices have been added to the system.

[FLOW CHART B]: The same startup flow with Redis changes incorporated.

[1] The system interrogates Redis to retrieve the current SCL file contents that represents the SMs in the system.

[2] The SCL file is validated to ensure that it conforms to the IEC 61850 standard.

[3] The Server starts with the current SCL file contents as the configuration.

[4] The Server interrogated once more, to validate if any new SMs have been added to the system.

[4.1.1] The server is stopped.

[4.1.1] The server retrieves the new SCL file contents, going back to step [1] restarting the startup process.

[5] The server reads the inputs provided.

The following section present the SM API's implementation. A discussion regarding the various integration points is presented.

4.4 Smart Meter API

The Smart Meter REST API is the centre of the system. The SMs communicate data with the API and keeps track of all the SMs that is online. The API contains a set list of http endpoints which are used to:

- Create a SM
- Submit a reading
- Get all readings
- Get a reading for a certain specific SM

To achieve the storage and retrieval the smart meter API makes use of the MySQL database to contain the information the system requires to function. The database technology being utilized is SQL Server. Like all other technologies utilized in this research project, Docker is used to run the server on the server-Raspberry Pi.

The system makes use of a .Net web API framework to create http endpoints to allow the transfer of data amongst the various servers and SMs. The database is interfaced with using a Microsoft technology, called Entity Framework.

4.4.1 Create smart meter endpoint

The smart meter API has various capabilities. The first capability is the ability to assign a SM a token, which is used to identify the SM. Each token is unique and cannot be replicated. This token allows the SM to identify itself to the system. Figure 4.15 shows a sequence diagram of the GET POST /Meter endpoint.

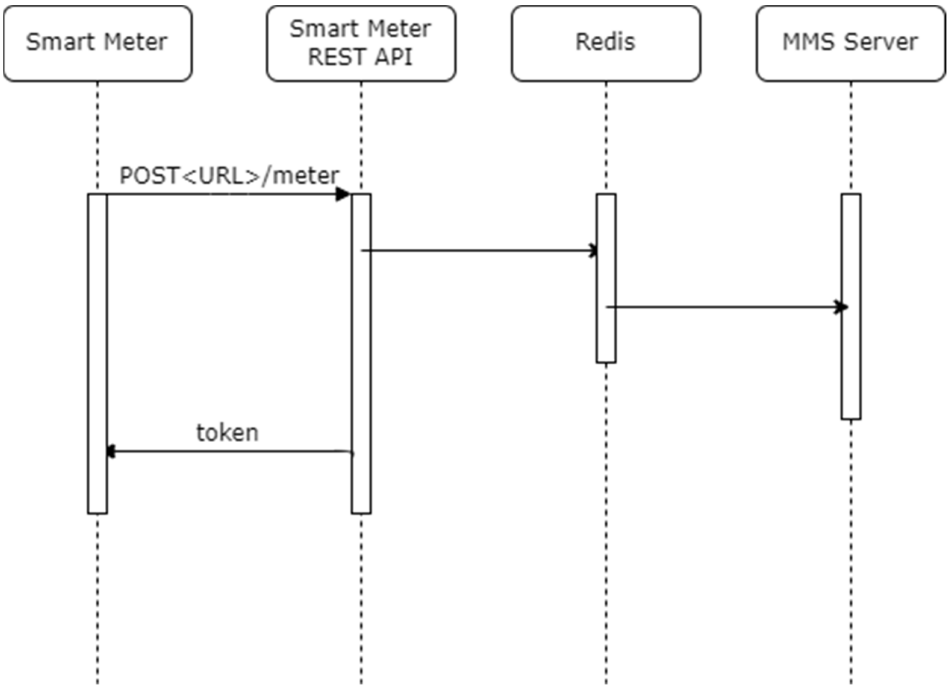

Figure 4.14: Sequence diagram of the create-meter endpoint

When a meter is created the structure of the system changes. To ensure dynamicity the MMS server is required to represent this new meter. To ensure this change happens the API publishes an updated SCL file to the Redis server. The MMS server then interrogates the Redis server to retrieve the latest system contents.

The POST meter endpoint requires a request body to be able to create a meter. The body consist of three attributes:

- Address Code
- IP Address
- Owner

These values are stored in the Smart Meter API's database to create a mapping of the meters, and their various readings Submit SM reading endpoint

Figure 4.16 shows the sequence required for a SM to submit data to the Smart Meter API. After the meter has been created and a token is retrieved by the SM. The SM uses the token to identify itself when submitting data to the API. The token becomes part of the URL endpoint for example:

POST https://api.metering.com/readings/**2c932263-420e-4f88-b495-0a016cafb046**

The *submit readings* call requires a request body to be submitted with the call. The request body contains the various readings the SM has recorded. The payload consists of a raw JSON form with the values for voltage, current, power and power factor.

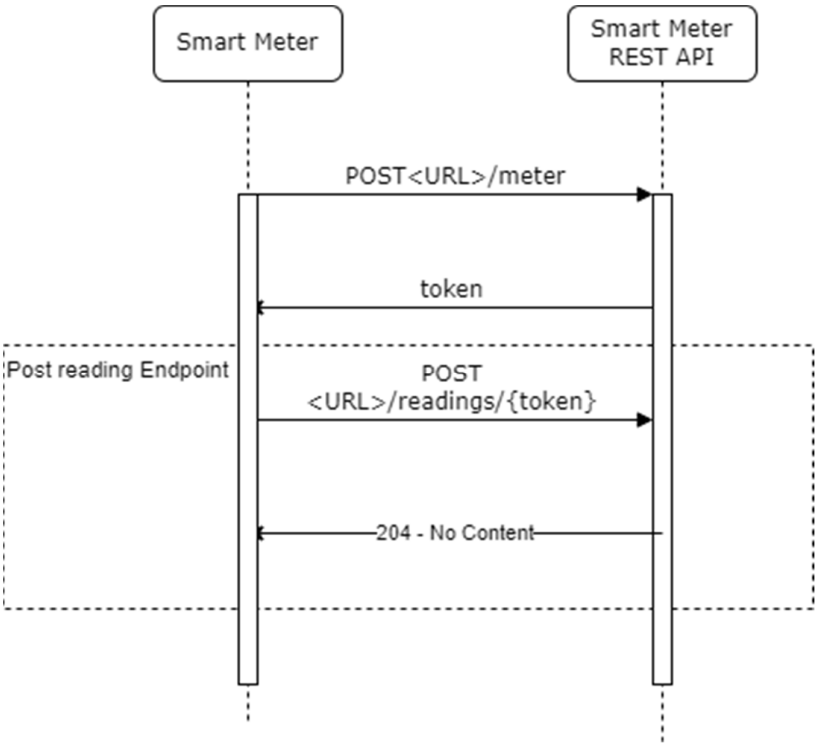

Figure 4.15: Submit SM reading API endpoint sequence diagram

4.4.2 Get all readings endpoint

Figure 4.16 shows the sequence at which the MMS server retrieves data from the Smart Meter API. After the SM has been created, and the SM has started submitting readings to the API, the MMS server is able to get all readings.

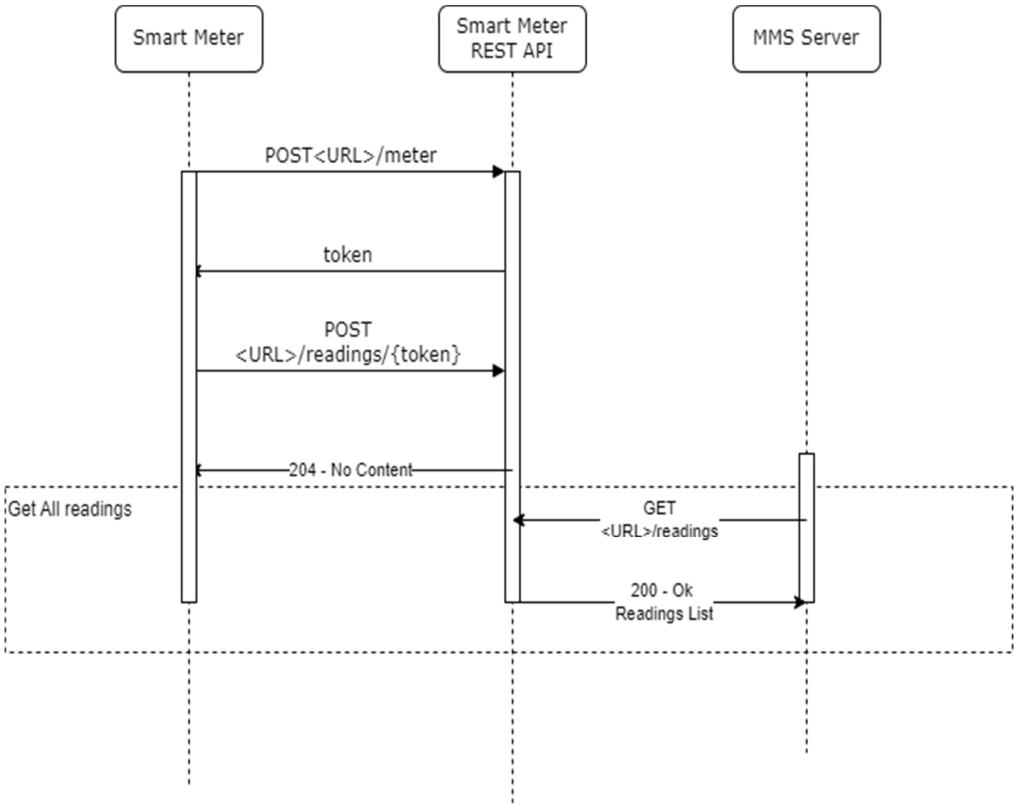

Figure 4.16: Retrieve SM readings API endpoint sequence diagram

4.4.3 Get readings for a specific smart meter

Another GET endpoint is available for the retrieval of a specific SM's data. The sequence at which this action takes place is the same as the GET all readings endpoint. The sequence diagram is in Figure 4.16.

This endpoint retrieves a single endpoint's readings. The device token is used as part of the URL to identify the meter in question:

GET https://api.metering.com/readings/**2c932263-420e-4f88-b495-0a016cafb046**

4.4.4 SCL file generation

Section 4.3 outlines the introduction to Redis into the system to allow for the efficient communication of the SCL files to the software on the two servers. The Smart Meter API is responsible for publishing the SCL file to Redis based on the SMs in its database.

The following section discusses the SM implementation along with its various hardware modules and software.

4.5 Smart meter

The SM is the target audience of this research project. The entire system is built around keeping track of metering data and making it available to various software's or IEC 61850 MMS clients.

The smart has two main components. The SM, and the module that is metering. The SM as previously discussed is a Raspberry Pi. The metering is done by a metering module called, PZEM-004T. The module contains a voltage and current transformer, this allows it to correctly read and calculate various values that would be of use to a metering device.

4.5.1 Python smart meter client

The SM Python software client is used to manage and send power usage data to the Smart Meter API. The Python client runs on the SM Raspberry Pi. The Raspberry operating system that was discusses earlier, already has Python software installed which allows software to just be executed.

It is assumed for the SM client, that the device always has access to a Wi-Fi network, which is connected to the Internet. The Internet connection allows it to communicate with the Smart Meter API.

The Python smart meter client contains a config file with all the relevant information to allow it to complete its function. The config file requires information:

- Wi-Fi SSID (Service Set Identifier)
- Wi-Fi password
- SM token
- Address Code
- Owner

This information is being utilized throughout the entire script. Figure 4.18 shows a flow chart of the smart-meter client's code.

[SM CLIENT – FLOW CHART]

[1] The client reads the config file.
[2] The client attempts to connect to the Wi-Fi using the details specified in the config file.
[3] The client validates if the device has connected to the Wi-Fi, if not step 2 is executed again.
[4] The client checks if a token is present in the config file.
[5.1] If the token is not present, make a POST API call to <URL>/meter/ with a body containing, *owner name, IP address and Address code.*

[5.2] Save response token to the config file of the SM client.

[6.1] If the token is present in the config file, retrieve reading value from the PZEM-004T power reading module.

[6.2] Make a POST <URL>/readings/**{token}** call to the smart meter API with the request body containing voltage, current, power and power factor, which is retrieved from the meter reading module.

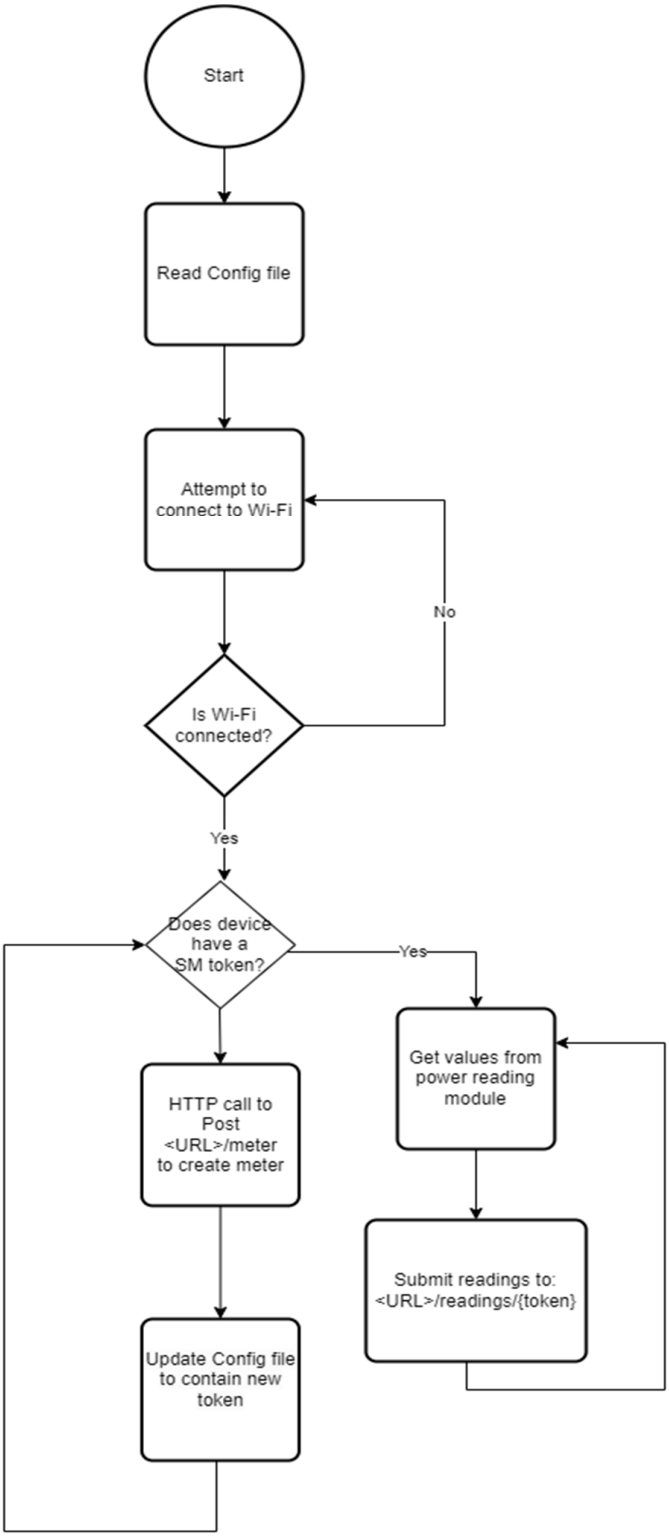

Figure 4.17 Flow chart of Python SM client software

4.5.2 PZEM-004T

The power reading module is used in two ways, with GPIO pins or a USB serial interface. The Python client makes use of various packages to import functionalities into its ecosystem. Amongst one of the packages is the *modbus_tk* package which contains driver software for the PZEM-004T module. This module connects to the module via a USB serial interface. Making use of the USB port is therefore crucial for the functioning of this module of the client ecosystem.

4.5.3 Smart Meter Simulator

A SM simulator is introduced to play the role of the SM. The Simulator consists mainly of a Python script that is run on any device that has Python software installed, this includes a computer or a Docker container or a Raspberry Pi. The simulator works in the exact same way as the physical SM would, except when interrogating the PZEM-004T module, the values being returned are randomized. Current, Voltage, Power, Power Factor and Frequency are generating numbers that fall into valid ranges.

4.6 Conclusion

This chapter detailed the implementation of the entire SM ecosystem. The online server implemented a Docker containerization software to allow it to quickly run the server software i.e. MMS Server and the Smart Meter API. A performance issue discovered in the implementation phase resulted in files that grew bigger impacting the performance of the API. The Redis caching software is introduced to solve this problem.

The MMS server software is found to be static, creating a stale server that never updates the devices present. The code is changed to accommodate a more dynamic flow of adding devices to the MMS server.

The SM had most of its implementation completed in the planning phase of this research work. A config file is introduced to better deploy the SM software. In addition, a SM simulator is created to allow more SMs to submit data to the API.

The following chapter presents the test and results phase of the SDLC. The implemented system presented in this chapter is used to test various scenarios and determine if the system can reach its goals. In addition, if tests are not able to be completed, necessary adjustments are made to accommodate the scenario.

CHAPTER 5

RESULTS AND TESTING

5.1 Introduction

This chapter presents the testing and results phase of the SDLC (Software Development Life Cycle). The previous chapter presented a partial implementation of a system capable of adding Smart Meters (SM) dynamically. This chapter details the testing of the solution and documents the findings.

In the SDLC the actor testing the system, creates test cases which are used to document the results of the tests for a particular part of the system. This section follows this format by declaring the test cases and a subsection for executing those cases. The results from executing the test are also documented along with the procedural steps that were taken.

This chapter consists of 7 sections. Section 5.1 introduces the reader to the background of the chapter and structure of the chapter. Section 5.2 outlines test cases the system is subjected to ensure that all parts of the system work correctly. Section 5.3 - 5.6 documents the execution and results of test cases outlined in Section 5.2. Section 5.7 concludes the chapter.

The testing of each component is done individually, adding to the system and testing individual components until the entire system is assembled. First the API is tested, then the SM is added to the system, testing the Application Programming Interface (API) and SM. Then the Message Manufacturing Specification (MMS) server is introduced with testing of the API, SM and MMS Server end-to-end.

5.2 Test Cases outline

Every system makes use of the SDLC implementation of the testing phase at the end of the cycle. The testing phase is documented by making use of test cases. Test cases are structured documented steps to execute a test and acceptance criteria to determine that the test is a success. This first section provides a high-level overview of all the test cases for each of the components. In the following section the test cases are executed in a step-by-step fashion.

End-to-end testing is a test that validates that the input in one part of the system provides a valid output in another part of the system. An end-to-end test validates the complete integration of the entire system.

All test cases are structured using three criteria: preconditions, test steps and acceptance criteria. Preconditions are events that need to happen before the test steps are executed. Examples of a precondition are starting the server, downloading the required software, or starting a client. Test steps are the execution of the actual tests and acceptance criteria are the successful conditions that validate the tests. If the acceptance criteria are not met, the test is marked as a failure. There are four parts of the system that require validation. These parts are categorized into their own test case scenarios.

- SM
- Smart Meter API
- MMS Server
- SCL Files

5.2.1 Smart Meter API

The smart meter API is tested first, due to the ability to exist without dependency on any other parts of the system. The Smart meter API requires a database to be able to successfully deploy itself. Hence the only prerequisite for the API is the database to be operational and running. The Database however requires a database schema to be applied to the database to be able to retrieve and store data.

Table 5.1: Component Test Case for Smart Meter API

Scenario	Success
Preconditions	1) Database is created 2) Migration script is ran against the database.
Swagger documentation page is accessible	Swagger page is displayed, after navigating to <API_URL>/swagger. All endpoints are displayed.
SM is created via the POST /SmartMeter endpoint	Using the GET /SmartMeters/ endpoint, returns a list where the new smart meter should be present
Readings are submitted with a valid API token using the POST /readings/{token} endpoint type.	Readings is in the list returned when making a call to GET /SmartMeter

The Smart Meter API's main functionality is tested by submitting readings to the API and asserting those values by requesting them back again.

The POST /SmartMeter endpoint is asserted by the GET /SmartMeter/{id} endpoint. Making use of the token retrieved from the call, verifies the existence of the newly added SM.

5.2.2 Smart Meter

The SM has three parts that requires proper testing. The test cases are based around the power reading module, the Wi-Fi connection and the firmware.

Table 5.2 shows the various scenarios; the SM requires to establish that it is in a functional state. It is seen that the top row contains the preconditions to be fulfilled for testing to commence. The four pre-conditions set up the SM to work in the environment it is placed. There are 4 negative test cases in Table 5.2 to validate the behaviour of the SM if one of the above-mentioned components are not functional.

Table 5.2: Component Tests for the Physical Smart Meter Unit.

Scenario	Success	Test Type
Preconditions for successful start-up.	1) The Python client is installed on metering device. 2) Smart Meter API is online. 3) Smart Meter is constructed with all power reading modules. 4) Smart Meter has access to Wi-Fi.	
Smart meter is not connected to any power source.	The smart meter reports 0 watts reading from the module.	Negative Test
Smart meter is connected to any power source.	The smart meter reports > 0 watts returned.	Negative Test
Smart meter is not connected to Wi-Fi.	Error message is reported, indicating that no networks are available	Negative Test
Smart meter is connected to Wi-Fi and displaying information.	Successful data request is reported to the Smart Meter API and a token file is created.	Positive Test

The test case presented in Table 5.2 tests various parts of the SM. The SMs power reading module is tested by determining if false reports of power consumption are being

recorded. This is tested in the first two scenarios after the preconditions have been completed. The following test cases tests the network connectivity to determine if the Wi-Fi connection allows the meter to connect to other parts of the system. The final section being tested, is an end-to-end testing where the entirety of the SM asserting the transfer of data between the meter and the API is tested.

5.2.3 MMS Server

Table 5.3: Component Test for the MMS Server

Scenario	Success
Preconditions	1) Redis Server running 2) Redis contains SCL_CURRENT Keyword
Redis contains the correct information	The Smart Meters returned from the API correlates to that of which is contained in the Redis server.
MMS Server is running	The MMS Client is able to load all the smart meters.
MMS Server data validation	The MMS Server correctly displays the meter that is contained in the API.

Table 5.3 outlines the test cases, to validate the functionality of the MMS Server. The MMS server uses the Redis software platform to temporarily store the SCL file contents. The Smart meter API publishes to Redis every time a mutation action happens or if the GET/SCL endpoint is executed.

5.2.4 SCL File

SCL type files are files that describe the substation and the elements thereof in a OOP oriented fashion. Amongst these are ICD files and SCD files, ICD files contain the functionality of a very specific IED devices. In our case the IED role is played by the SM. The SCD however is the combination of not only all devices in the substation but also various details about the substation like networks and metadata.

In this case the entire SCD file is amended every time a new SM is added. The addition being made is the contents that would be typically found in an IED file. Appendix 8.2 shows an example of a completed SCD file contain various SMs. In the file the contents of the *IED* tag, represents the SM.

The SCL test cases consists mainly of file validation to ensure that the files are readable by various software platforms and the MMS Server based on the IEC 61850 standard.

Table 5.4: Component tests for the SCL file

Scenario	Success
Preconditions	System has smart meters active
XML Validation	XML Utility tool can read the SCL file.
SCD Validation	SCL configuration tool can read the SCD file and identify the requisite hierarchical structure specified by the IEC 61850 standard.

In the following sections, the test sequences outlined in tables 5.1 – 5.3 above are executed. Each table is tested in its own respective section.

5.2.5 End-to-End Testing

After all components are tested, an end-to-end test is conducted to ensure that the behaviour of the integrated system components is as expected. This test is done by integrating the entire system components together. Each component interfaces with the rest of the system in a predefined manner. The main end-to-end test is outlined in Table 5.5.

Table 5.5: Test steps for end-to-end testing and validation

Step description	Step outline
Preconditions	All system components are online, i.e. Redis, SM API and MMS Server.
Start new Smart Meter	A Smart Meter is turned on in a network with the appropriate network connection.
Validate API	Retrieve smart meter information and readings from the API, making use of the SM token.
Validate MMS Server	MMS Server is updated with the latest smart meter name.

5.3 Testing - Smart Meter API

5.3.1 Test Case 1

Testing the API is done via the .NET documentation software, Swagger. The Swagger instance is accessible by navigating to the API address. Making use of Table 5.1, we navigate to the Smart Meter API. The documentation is located on the *http://localhost/swagger*. A list of all the available endpoints is displayed, as seen in Figure 5.1. Running the SM-API code will generate the HTML document below which allows us to test each endpoint manually. There are multiple tools that exist that are used to test API calls. The industry standard for APIs is the OPENAPI standard, which outlines the software tool *Swagger*. Each endpoint is constructed with its contract (required JSON object). The contract being the payload that is sent to the server in the request body. In this case the request body takes the form of a JSON formatted payload.

This first test is to determine if the API server launches as per the document in the browser. The following steps depicts the actions taken.

5.3.1.1 Test steps

Step 1: Start API software – this is done by either clicking the run button in any .NET IDE (Integrated Development Environment) or in any command line the command “dotnet run” is executed.

Step 2: Once step 1 is completed, the default browser should be displayed navigating to the Swagger page which is seen in Figure 5.1.

Acceptance Criteria: This test is considered successful as soon as the page is automatically opened. The significance of the page opening is that a successful compile and startup is required for this event to occur.

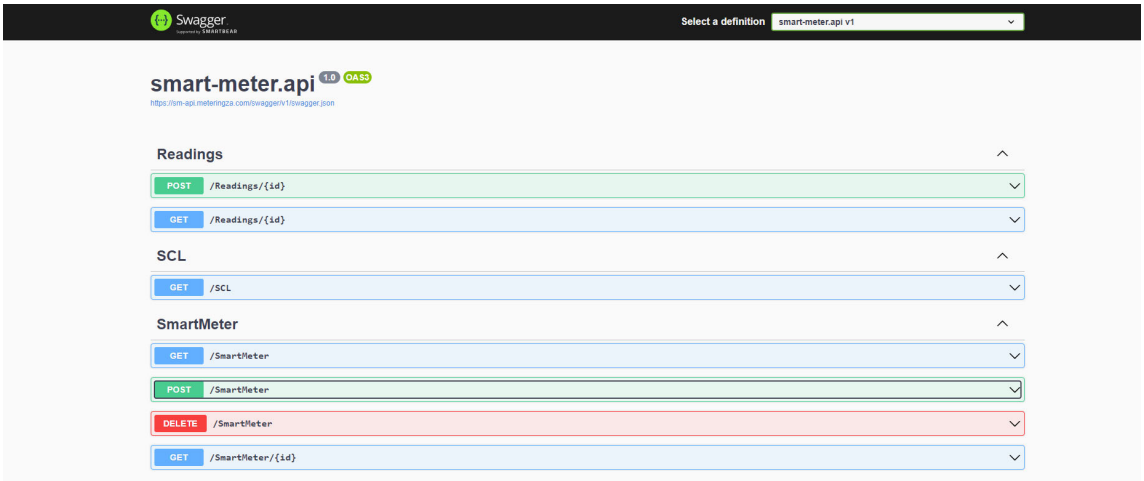

Figure 5.1: Generated Interactive API documentation

5.3.2 Test Case 2

The second test case is to determine, that SMs can be created on the API. Figure 5.2 shows the POST /SmartMeter call being made. This will ensure the existence of the digital SM with the GET /SmartMeter. The response from this endpoint is a payload containing an Id along with the information entered. The Id is used as the SM token to identify the meter. This test not only tests the POST /SmartMeter endpoint, but also tests the database that the API needs.

5.3.2.1 Test steps

The test steps follow a flow of creating a SM in the API using the POST call. Since the SM is not implemented yet, a Global Unique Identifier (GUID) is required to test the endpoint. The GUID represents a value that identifies the SM.

Pre-conditions:

1) Start the API and wait for the Swagger page to generate.

Test steps:

1) Open the **POST /SmartMeter** endpoint on the Swagger page. Create a body with valid test data for the contract. Three values are required: “name”,”location” and “AddressCode/ZipCode”.
2) Make a request by pressing the “Execute” option.
3) The Response Body should contain the same input data with the addition of the “ID”. Figure 5.2 shows the result of running the **POST /SmartMeter** endpoint.

4) Make an additional API call to the **GET /SmartMeter/{Id}**, with the Id that is generated from the **POST /SmartMeter**.

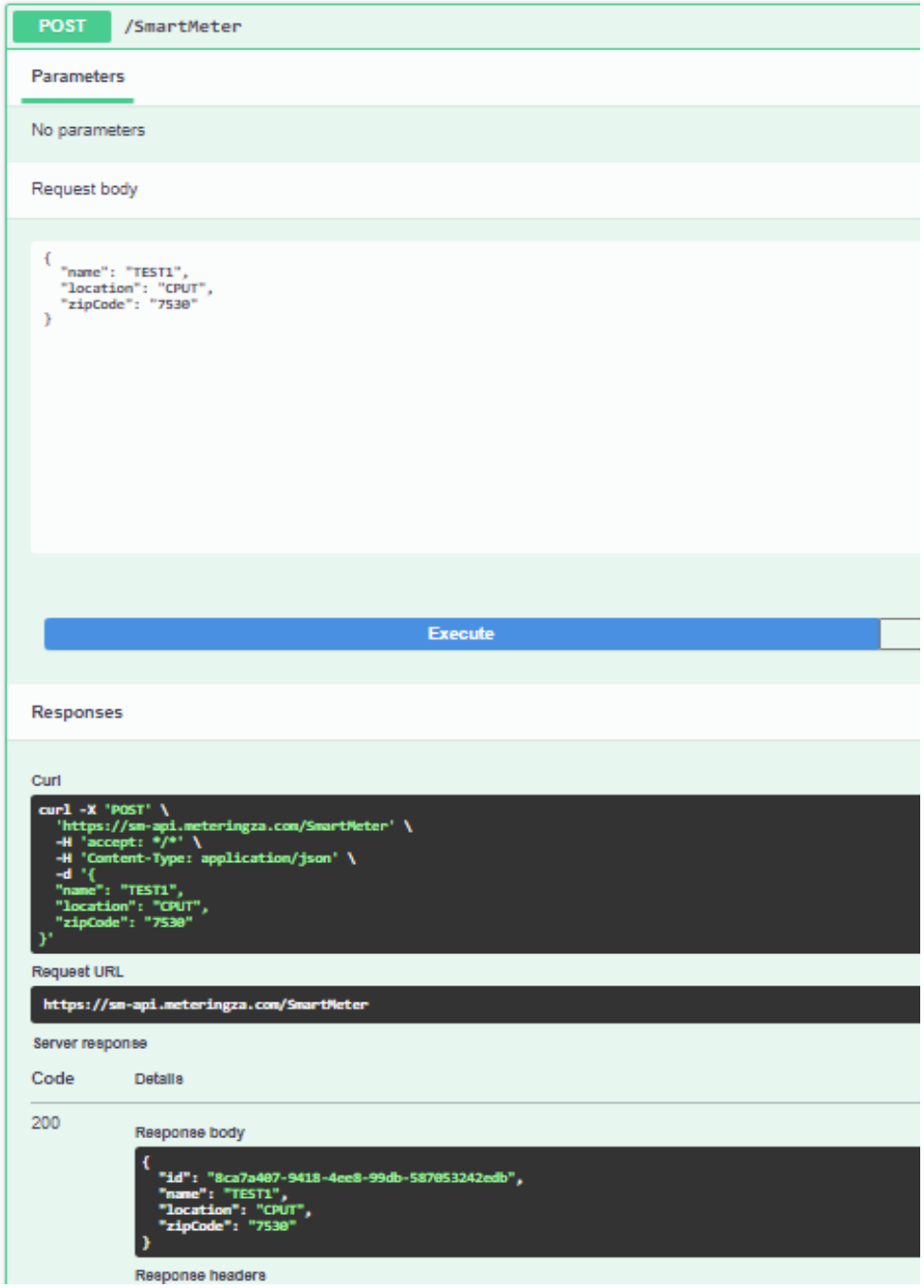

Figure 5.2: Swagger indicating successful API request

The token is then used to ensure that the SM is in the ecosystem by making another API call to **GET /SmartMeter/{id}**. The Figure 5.3 shows that 204 is being returned

which means the SM does exist within the system. If 404 is displayed, the meter wasn't located.

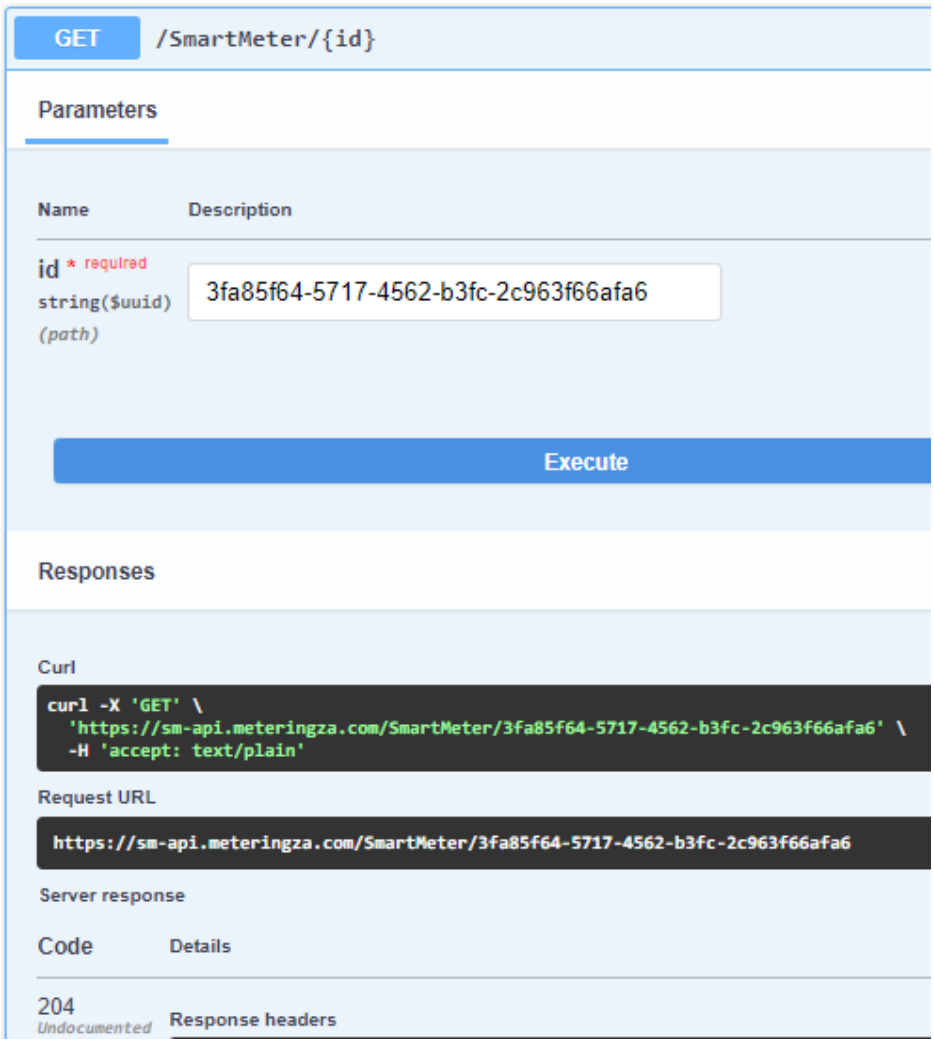

Figure 5.3: Swagger indicating successful GET request

5.3.3 Test Case 3

The following test case validates that the reading being submitted to the API, can be retrieved. The **GET /Readings/{id}** endpoint retrieves all the readings related to the token it has been provided with. This endpoint provides a method of retrieving a single reading for an already registered meter. This endpoint is dependent on the **POST /reading** endpoint.

5.3.3.1 Test Steps

Pre-conditions:

1) Start the API and wait for the Swagger page to generate the API document onto the browser.

2) Open the **POST /SmartMeter** endpoint on the Swagger page. Create a body with valid test data for the contract. Three values are required: “name”,”location” and “Address/ZipCode”.
3) Extract the new SM *Id* which should be in the form of a GUID.

Test steps:

4) Make a request to **POST /Reading/{GUID}** endpoint to submit a reading. Making use of the endpoint contract. Four values are required: voltage, current, power and power factor. The header parameter *id*, is required to identify the SM. This would be the GUID extracted in step 3.
5) Execute the endpoint.

Validation:

6) The response after executing step 5, should be a response code of 200 – indicating a successful HTTP response. The JSON body response includes all the submitted values along with a GUID that can be used to identify the JSON object. Figure 5.4 shows the request and the response of this validation.
7) Figure 5.5 shows the result of the **GET /Readings/{id}** which is used to retrieve all the readings submitted for certain SMs. The reading submitted in step 6 will appear as a single entry in this JSON list representing all the readings submitted for this SM.

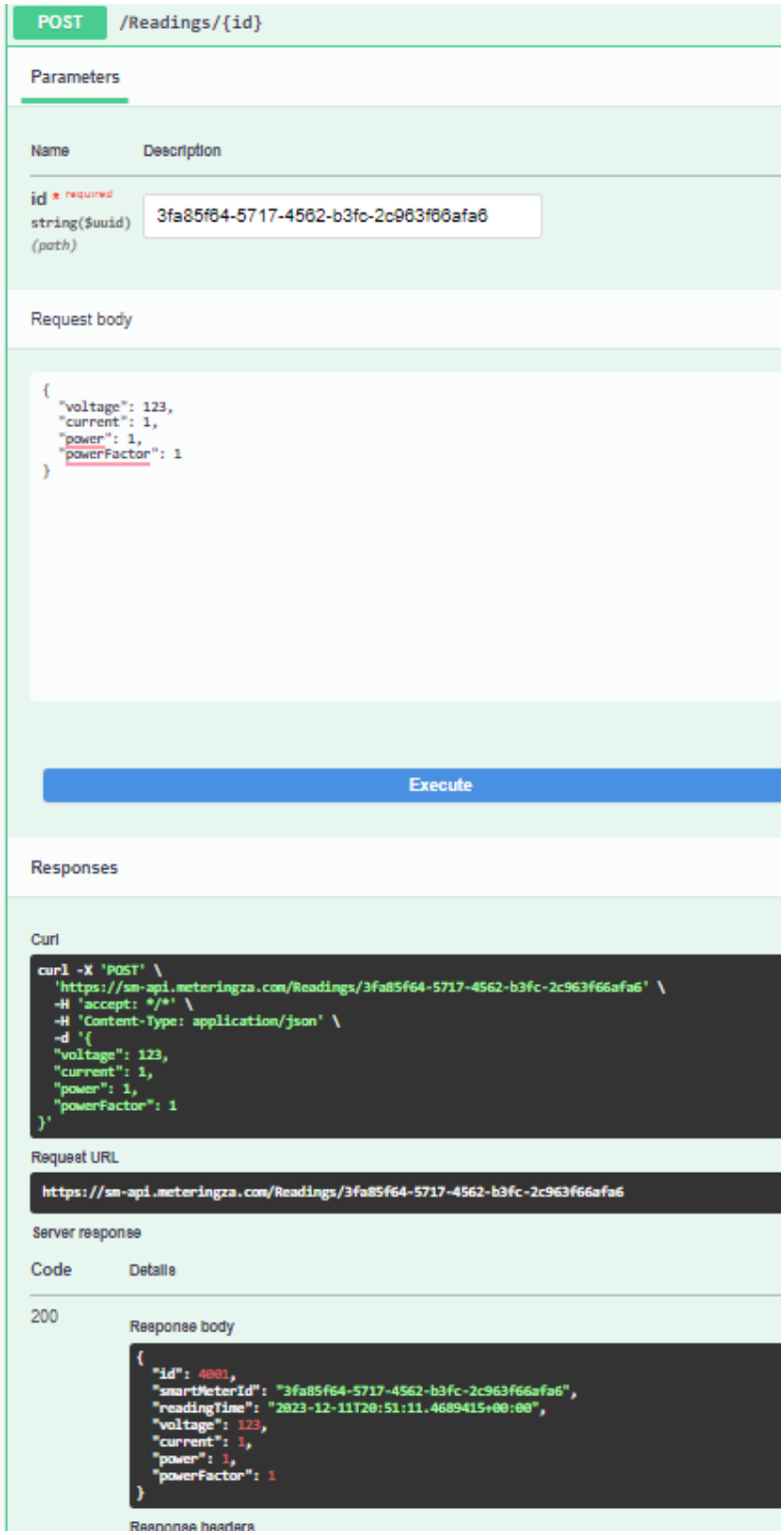

Figure 5.4: Swagger indicating successful GET power reading request

Each JSON entry contains seven attributes which can be seen in Figure 5.4. Voltage, Current, Power and Power Factor are some of the important values being recorded.

Figure 5.5 shows the formatted JSON payload. The expanded reading shows the same format as the response from the GET /Readings/{token} call. In addition, every other reading API call is also being returned in the JSON format.

This section presented the executed test cases in Table 5.6. All acceptance criteria are met; therefore, it is safe to say the Smart Meter API is functional.

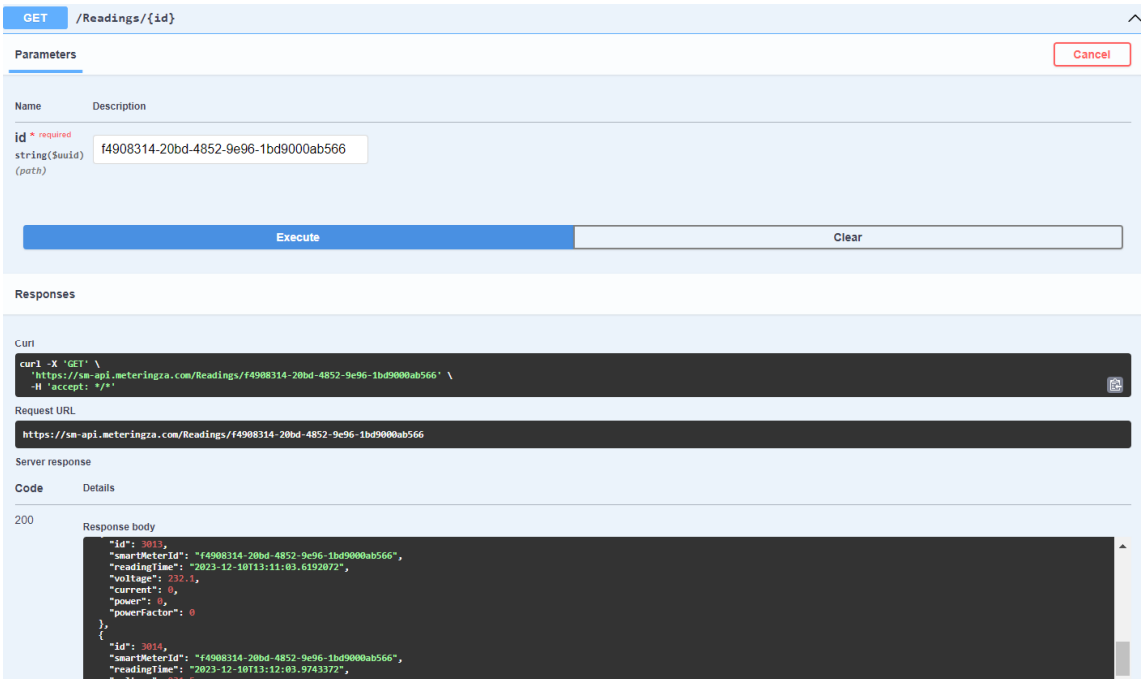

Figure 5.5: Swagger indicating successful reading retrieval

To ensure that the correct number of readings are retrieved, the JSON response is analysed. The reading date time are used to find the most recent addition to the meter's readings.

Table 5.6: Test Case results for the SM API component

Scenario	Acceptance Criteria	Acceptance Criteria Met
Preconditions		
Swagger documentation page is accessible	Swagger page is displayed, after navigating to <API_URL>/swagger. All endpoints are displayed.	Yes

Smart Meter is created via the POST /SmartMeter endpoint	Using the GET /SmartMeters/ endpoint, returns a list where the new smart meter is present	Yes
Reading can be submitted with a valid API token. Via the POST /readings/{token} endpoint	Readings is in the list returned when making a call to GET /SmartMeter	Yes

The following section presents the execution of test cases to test the SM.

5.4 Testing - Smart Meter

5.4.1 Test Context

The SM token is crucial to enable SMs to submit readings to the Smart Meter API. The SM is therefore dependent on the Smart Meter API. To allow the SM to be tested a valid instance of the API needs to be running. Therefore, no test data can be used to test the SM.

Running the SM firmware allows the retrieval of a SM token. The token is then stored in a file in the SM. The token is retrieved by logging into the SM via the Secure Shell (SSH) protocol and interrogating the directory where the meter API client is running. The research laboratory contains a Wi-Fi network which is used as the network under test. In Table 5.7, a list of IP-addresses is found. DNS names are given to the MMS server and the Smart Meter API.

Table 5.7: Network Addresses for all system components

Component	IP Address	Online URL (DNS)
Main Router	**192.168.100.1**	**N/A**
Gateway	192.168.100.1	**N/A**
Smart Meter	192.168.100.103	**N/A**
MMS Server	192.168.100.102: 102	mms.meteringza.com
Smart Meter API	192.168.100.102: 8901	sm-api.meteringza.com
Redis	192.168.100.102: 6379	**N/A**
Research Computer	192.168.100.111	**N/A**

It can be seen from Table 5.7 that the MMS Server, Redis and Smart Meter API are all running on the same server. Each of the server software contain their own port to enable interfacing. The servers are also hosted online to allow SMs outside the laboratory network to submit data.

Making use of the information in Table 5.7 a SSH connection is established with the meter. Running the command *ssh {username}@{ip-address}* creates a remote connection to a device. A prompt will immediately be displayed prompting the user for a password as shown in Figure 5.6.

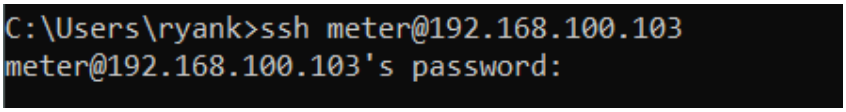

Figure 5.6: Results of running SSH command

The SM client is copied to the SM with the command ***smart-meter-client.py***. The requirement of the script is installed by *running pip install -r requirements.txt*.

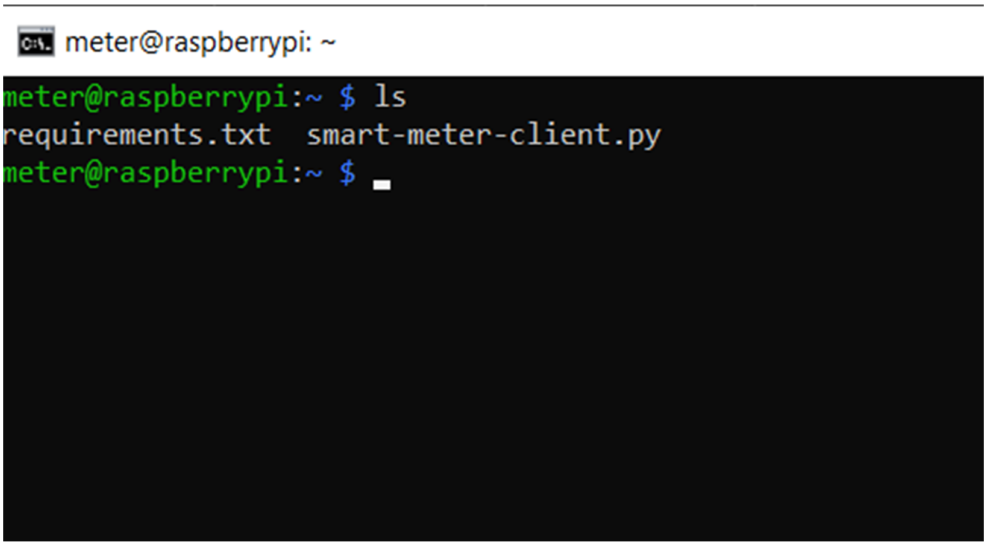

Figure 5.7: Results of directory command executed on the SM

The script is run using the ***python3 smart-meter-client.py.*** Figure 5.8 shows the output of the script: The token of the meter is retrieved by stopping the client. A new file has been created named: “token.txt”. The token.txt is created to contain the token the API has assigned to the SM. This is considered the permanent storage of the token for the device.

Figure 5.8: Result of running smart-meter-client.py

Since the SM meter is running on Raspberry Pi with a Linux operating system, various text editors can be used to read the token.txt file. The file is read by making use of “nano token.txt”.

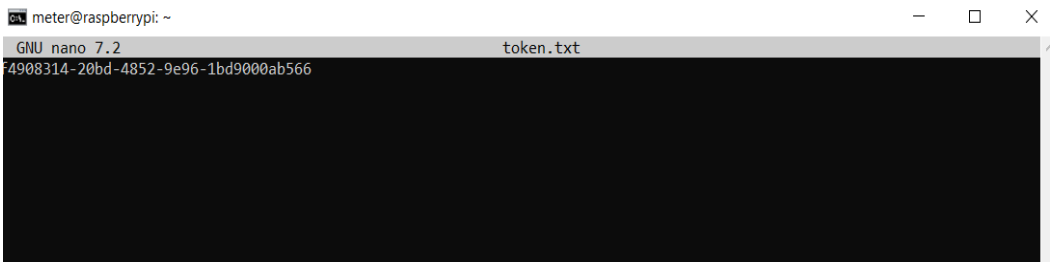

Figure 5.9: SM generated code

The resulting token is: ***f4908314-20bd-4852-9e96-1bd9000ab566***. This token can now be used to interact with some of the SM API endpoints. Figure 5.5 in this chapter shows the result of requesting readings from the SM created in Figure 5.9.

The testing of the SM can proceed once these steps are followed.

5.4.2 Test Case Execution: Smart Meter

All the following SM tests have the following preconditions:

Pre-conditions:

1) Start the API and wait for the Swagger page to generate.
2) Assemble the SM consisting of PZEM-004T to the Raspberry PI. Add the meter to the network.

5.4.2.1 Test steps

The first test defined in Table 5.2, is to test the power reading module of the Raspberry Pi board. The module should read a value of 0 watts if the power is disconnected since there is nothing drawing power. The input voltage should however still read +- 230V. This is validated in the following steps.

Test:

1) Connect the PZEM-004T in series to read the power of any circuit or mains power supply within a house.
2) Turn on the SM and make sure that it is connected to a network.
3) Retrieve the SM token similar to that as illustrated in Figure 5.6 and Figure 5.7.
4) Validate the power readings on the SM using the SM token and the SM API. Ensure that the module is reading a valid power reading.
5) Disconnect the SM from the circuit.

Validation:

6) Validate power readings after disconnection reported by the SM via the SM API – GET Power readings should report 0 watts power usage along with a constant input supply voltage of +-230 Volts.

Figure 5.5 shows the result of the **GET /Readings/{id}** which is used to retrieve all the readings submitted for certain SMs. The reading submitted in step 6 will appear as a single entry in the JSON list representing all the readings submitted for this SM.

5.5 Testing - MMS Server

5.5.1 Test Context

The MMS Server converts the Redis value of SCL_CURRENT, into a SCL file. This file is used as the substation configuration file in the MMS Server.

The MMS Server is started by running the commands shown in Table 5.8 below. It is necessary to set the JAVA_HOME environment variable because Gradle uses it to build the project. In the project folder the *build project* command should be executed to download the necessary dependencies and generate files required to run the project.

Once the build commands have been executed, a new folder, named *build* should be located in the project folder. In the */build/libs* folder *.jar* files should be visible; these are generated by Gradle.

The MMS client and MMS server can be executed by using the commands in Table 5.8. The main projects are the MMS Server, the MMS console client and the MMS GUI client. Using the GUI client is beneficial as all the SMs and their data can be visualized in a list.

The table also describes commands that are used to run different parts of the MMS project. The MMS server and the GUI MMS Client are used to run each one of their

projects respectively. The Java installation command is required to be executed before any of the programs can be started, because they require Java.

Table 5.8: Table showing the different commands to run the various MMS tools

Function	Command
Pre-requisite 1	sudo apt-get install default-java export JAVA_HOME=<PATH_TO_JAVA_INSTALLATION>
Pre-requisite 2	/libs/build/gradlew clean build –refresh-dependencies gradlew shadowJar
Console Client	Java -cp /build/libs/<jarfile> com.iec61850bean.app.ConsoleClient
GUI Client	Java -cp /build/libs/<jarfile> com.iec61850bean.app.GuiClient
MMS Server	Java -cp /build/libs/<<jarfile> com.iec61850bean.app.ConsoleServer

5.5.2 Preconditions

To fully execute the following test Redis and MMS related test cases various pre-conditions must be met. The Redis and MMS Server requires the SM API to be active to receive updates about the system changes. The SM API is started with a mock SM to begin the testing with.

5.5.3 Test Case Execution: Redis Server

Testing Redis is crucial to the system, since the Redis instance is the asynchronous instance that keeps the system updated. The SM API updates the Redis instance every time something changes, and this allows the MMS Server to retrieve the latest changes and makes sure that it is consistent with the rest of the system.

This test case is to ensure that the Redis instance changes every time a SM is added. Making use of the **POST /SmartMeter** endpoint on the Swagger instance a new meter is added to validate if the **SCL_CURRENT** value changes. Multiple meters are added, and the size of **SCL_CURRENT** is then validated to ensure that the new meters are being added.

The Redis client can be used, but for a more graphical view of the object, *Redis Insight* can be used. To view the Redis cache a tool is used to connect to the instance to view

its contents. Figure 5.10 below shows the current contents of the SCL_CURRENT key. An additional tool is used to look at the size and the structure of the XML document.

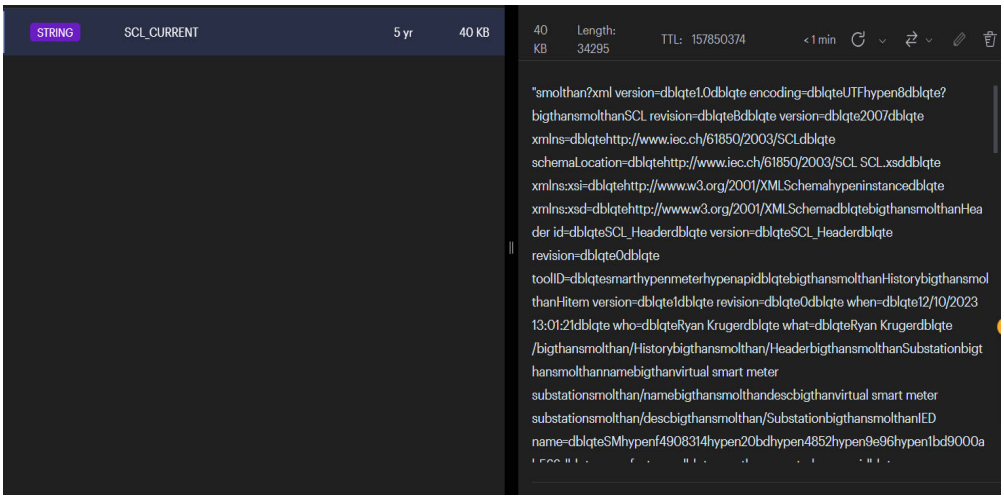

Figure 5.10: Redis interface showing the value of SCL_CURRENT

Steps:

1) Create a new SM using the POST /SmartMeter API.
2) Validate the SCL_CURRENT object in Redis by connecting to <SERVER_IP_ADDRESS>:6379
3) An XML viewer or notepad can be used to view the object.
4) The before and after of the object size can be ascertained to determine if a new SM was added. Table 5.9 show the changes in the sizes of the object as SMs are added.

Validation:

5) To ensure more meters are added SMs are gradually added to ensure that the size is also increasing.

As more meters are created using the: POST /SmartMeter endpoint the size increases linearly. Table 5.9 shows the size of the SCL_CURRENT value increasing as the amount of SMs increase.

Table 5.9: Size of the SCL_CURRENT object as more SMs are introduced

Smart Meters	Size of Payload (Kilobytes)
6	40
8	48
39	112
45	128

5.5.4 Test Case Execution: MMS Server

Test Context:

The MMS repository contains a GUI client, to allow the researcher to view the contents of the MMS Server with ease. The GUI client can be started by executing the command in Table 5.8.

Figure 5.11 shows the MMS GUI client. The client connects to the MMS Server by making use of the IP Address and port on which the MMS server is hosted. Making use of Table 5.7 and the port listed.

Instead of using human readable words to describe the logical device, a notation is used to describe the SM. SM-*{GUID-ID-TOKEN},* is used as the name for the logical device. This allows the SM name to be unique, since multiple SMs are added to the list of SMs. From the list the token that is created for the SM is visible and all logical nodes are displayed.

Figure 5.11 also shows the various logical nodes that are associated to each SM. It can be seen the figure shows the context of a singular SM. Each SM can be seen containing the logical nodes:

LLN0 – Short for logical node 0, this is a compulsory logical node that is used to describe that device and attributes of communication.

LPHD1 – Logical node representing the physical device.

MMXN1 – This is a singular phase meter logical node. Mainly used for recording power characteristics like current, voltage, power and power factor.

CSWI1 – This is a control logical node that normally controls a switch or multiple switches.

XSWI1 – This is the logical node that represents the circuit switch in the system.

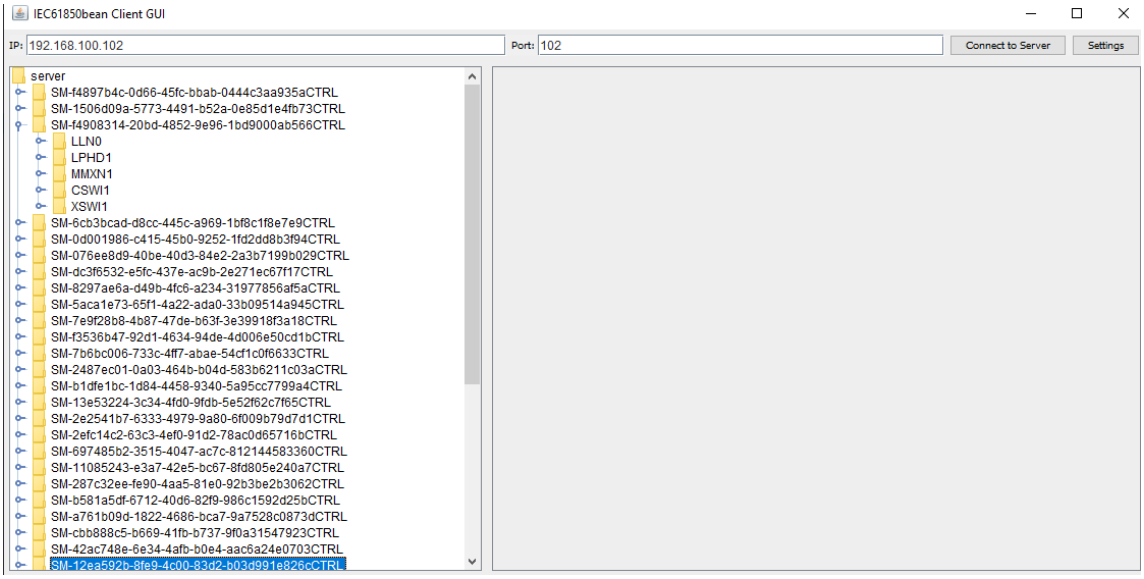

Figure 5.11: Hierarchy of SCL system from the perspective of the MMS server

The following section executes tests to validate the useability of the generated SCL file in an IEC 61850 environment using IEC 61850 configuration tools.

5.6 Testing - SCL File Validation

The IEC 61850 standard makes use of SCL files to describe a substation as described in part 6 of the standard. The implemented system dynamically updates the SCL file. Ensuring that the SCL file is valid is crucial, as this is the source of information to the MMS Server. A list of tools has been gathered to test various aspects of the generated SCL file. Below a table can be found containing the tools used to complete the testing of the SCL file.

Table 5.10: Tool selection for validating the validity of the SCL file

Tool	Was the file readable	Warnings	Error Messages
sclwebcheck	Yes	Yes	Yes
XML Marker	Yes	No	No

SCL Navigator	Yes	No	No
ICD Designer	Yes	No	No

5.6.1 Test Context

The SCL file is dependent on at least one SM being present in the system. As soon as the first SM is created the initial SCL file is also generated. As more SM are added or deleted the SCL file will update.

The SCL file can be found in multiple places. The main place where the file is stored is in Redis in the “SCL_CURRENT” keyword. The value can be copied to a .SCL file and fed to the various tools that requires it. The other place it can be found is in the Smart Meter API where the repository has been created. The file is named “virtual-substation.scd”.

5.6.2 Test Case execution: SCL file Formatting

This test case asserts the readability of the SCL file. The SCL file is in XML format therefore it is important to ensure the XML structure is not compromised, since the entire file is being generated by the Smart Meter API. Making use of various tools, all test cases can be satisfied.

Table 5.10 shows all the tools that were used to ensure the SCL file’s validity. Amongst the tools is XML marker that is used to ensure that XML file conforms to the correct file structure.

Figure 5.12: Tool *sclwebcheck* results for reading SCL_CURRENT Object

Figure 5.12 shows the result of the XML Marker software reading the SCL file output from the Smart Meter API. It is seen from the figure that the software reads the entire file correctly and can draw a hierarchical depiction of the structure of the file.

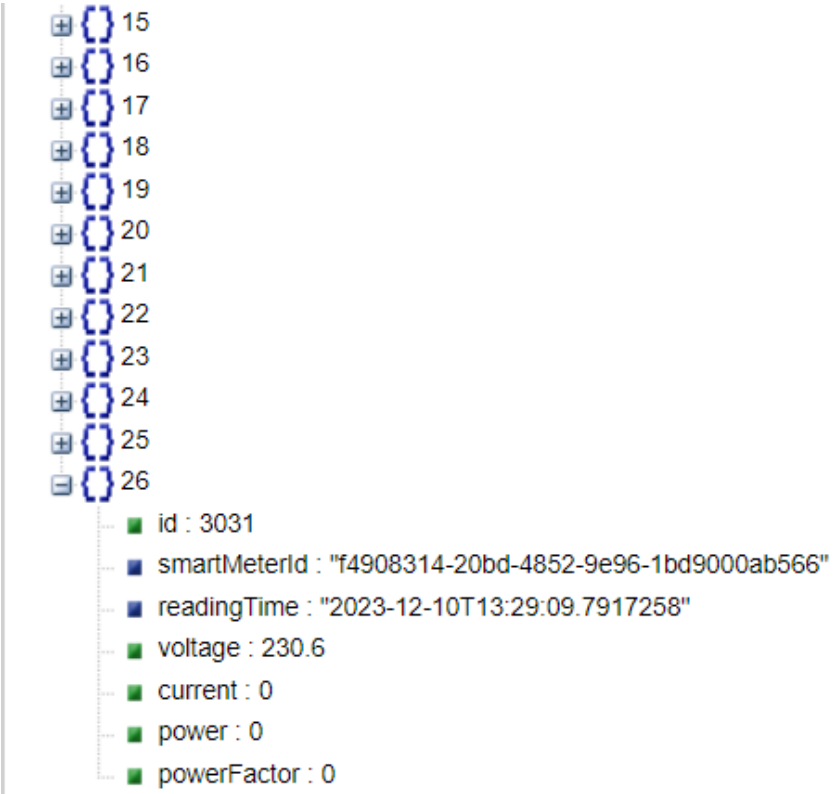

Figure 5.13: IEC 61850 approved tool reading generated SCL file

The XML Marker software does not make use of any IEC 61850 standard validation. The next section tests the file with IEC 61850 standard rules.

5.6.3 Test case execution: IEC 61850 Validation test

The MMS Server is designed to be used in an IEC 61850 environment. The SCL file is the part of the standard that outlines the hierarchical depiction of a virtual substation. This test case validates the IEC 61850 compatibility of the SCL file against approved IEC 61805 configuration tools.

The first tool is *sclwebcheck*, which is an online tool. This tool requires that a SCL file be uploaded to a website. The software validates the file and provides an output based on what was read. If the file is not readable, the software will give the appropriate output.

Figure 5.14 shows the output of various tests the website runs against the SCL file. There are various warnings and errors. Reported warnings and errors returned from this tool are valid. The main goal of the SCL file is to construct a file readable by any IEC 61850 system.

```
-------------------------------------

ALL MESSAGES BY LINE ***
---------------------------------------
        TYPE    LINE    MESSAGE TEXT
        Warning         Found "ConfReportControl" in Services section but failed to find any Report Control Blocks
        Warning         Found "GOOSE" in Services section but failed to find any GOOSE Control Blocks
        Warning         Found "ConfLogControl" in Services section but failed to find any Log Control Blocks
        Warning         Found "ConfReportControl" in Services section but failed to find any Report Control Blocks
        Warning         No Communication section found
        Warning         No Communication section found
        Warning         No Communication section found
        Warning         Unused data template DOType id=TMW_Generated_DPC.
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrSupAlm.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrDn.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrUp.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element InOv.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element Proxy.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element OutOv.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element Loc.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element LocKey.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrSupAlm.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrDn.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element PwrUp.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element InOv.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element Proxy.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element OutOv.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element Loc.stVal
        Error           "bType" mismatch.  Master bType of BOOLEAN != INT32 for element LocKey.stVal
        Warning         Found "ConfLogControl" in Services section but failed to find any Log Control Blocks
        Warning         Found "GOOSE" in Services section but failed to find any GOOSE Control Blocks
```

Figure 5.14 Output from *.icdDesigner* outlining various errors and warnings

The output from *sclweb* check, was able to read the SCL and found various issues and warnings regarding the file. The issues are very specific issues. “bType mismatch” was an intentional change in which the data type of an Integer attribute was changed to a Boolean data type. Two things can be deduced from this information. The software was

able to identify logical nodes and that the attributes of certain ones were incorrect. The software is designed in a way that the SCL file can be tested against various attributes or datasets.

5.7 Conclusion

The results and testing phase of this research provided critical validation of the proposed smart metering system, demonstrating its adherence to the IEC 61850 standard and its effectiveness in achieving the desired objectives. By conducting a series of structured test cases, the chapter evaluated the performance of the system in terms of end-to-end functionality.

The testing process confirmed the functionality of individual components, including the Smart Meter API, MMS server, and hardware integration with Raspberry Pi devices. Furthermore, the end-to-end tests validated the seamless communication between system components and ensured the accurate transmission and retrieval of energy data. These results highlight the robustness of the implemented design and its capacity to operate in distributed energy environments.

While the system performed well under controlled conditions, the chapter also identified areas for improvement, such as addressing network variability and enhancing security protocols. These insights provide a foundation for refining the system and exploring further applications in real-world scenarios.

In summary, the results and testing chapter established the feasibility and potential of the proposed smart metering system, bridging theoretical concepts with practical implementation. The findings reinforce the viability of integrating standardized communication protocols in advancing SG technologies and provide a strong basis for future development in this domain.

The following chapter provides the conclusion to this research work.

CHAPTER 6

CONCLUSION

6.1 Introduction

The rapid transformation of energy systems worldwide necessitates innovative approaches to ensure efficiency, reliability, and sustainability. This research aimed to bridge the gap between the IEC 61850 standard and modern technological advancements, focusing on Smart Meters (SM) as pivotal components within the Smart Grid (SG). By integrating the MMS protocol with contemporary IoT and web service technologies, the study proposed a scalable and interoperable solution for energy management and monitoring. Incorporating various cloud computing modules to allow the system to dynamically adapt to grid changes.

6.2 Thesis deliverables

6.2.1 Literature Review on MMS Protocol and the IEC 61850 Standard Uses in the Smart Grid

The literate review yielded the key findings from the literature on the MMS protocol and its application within the IEC 61850 standard. It demonstrated how the MMS protocol facilitates real-time, standardized communication, contributing to the interoperability and scalability of SG systems. The review confirmed the feasibility of extending IEC 61850-based solutions beyond traditional substation environments, providing a foundation for their application in residential SMs.

6.2.2 Literature Review on MMS Protocol and the IEC 61850 Standard Uses in the Smart Grid

The literature highlighted the critical role of web services in enhancing the functionality of IEC 61850-compliant systems. The literature review uncovered the advantages of integrating web-based architectures, such as REST APIs, to improve data accessibility and streamline system scalability. It also addressed challenges in ensuring compatibility between web services and the IEC 61850 standard, guiding the implementation of a robust and efficient communication framework for the proposed system. The conclusion of this part of the literature review indicated that web-based architectures open the possibility of containerization within the SG.

6.2.3 Literature Review on the Design and Applications of Residential Smart Meters

Insights on the evolution of residential SMs, focusing on their design principles, applications, and limitations were achieved. The review emphasized the need for

interoperability and identified opportunities for leveraging open-source technologies to achieve cost-effective yet high-performing smart metering solutions. These findings directly informed the design and deployment of the proposed system, ensuring its adaptability to residential energy environments while maintaining alignment with standardized protocols.

6.2.4 Analysis of the IEC 61850 MMS Protocol Applicability

The study demonstrated the applicability of the IEC 61850 MMS protocol in smart metering systems, revealing its significant advantages in enhancing communication within distributed energy environments. The protocol's ability to ensure seamless data exchange and interoperability among diverse devices was validated, making it a robust choice for modern energy management systems.

The lack of support for the MMS protocol created various challenges with finding secure up-to-date software that can be utilized for an open-source project. This resulted in the researcher changing an out-of-date MMS library to allow new functionality, albeit by using an unsecure library.

6.2.5 Exploration of Object-Oriented Capabilities

The research explored the object-oriented design principles embedded in the IEC 61850 standard, emphasizing how these capabilities contribute to achieving standardization in SG communication. Making use of the standards and OOP principles it was sufficient to recreate the models used to describe a substation in a OOP language such as .Net and Java.

6.2.6 Implications of Adopting International Standards

The study investigated the adoption of international standards like the IEC 61850 standard in achieving interoperability, scalability and reliability in smart metering systems, and by adding the use of cloud computing. The findings highlighted the benefits of centralized data management and processing, demonstrating how cloud-based solutions enhance system scalability and ensure consistent performance across distributed networks.

6.2.7 Implement a smart metering system with IEC 61850

The implementation successfully demonstrated the integration of the IEC 61850 MMS protocol into a smart metering system. The implementation made use of the Docker containerization software to encapsulate all technologies that are required for information processing from to end-to-end. Data storage issues are overcome by

making use of a Redis database to temporarily store the SCD file content instead of storing it permanently.

The system enabled seamless communication and standardized data exchange, meeting interoperability requirements for residential energy environments. This achievement validated the feasibility of applying IEC 61850-based solutions beyond traditional substation settings, addressing a critical gap in energy communication systems.

6.2.8 Utilize Open-Source Technologies for Cost-Effective Deployment

The use of open-source hardware and software, such as Raspberry Pi devices and Docker environments, proved effective for smart metering solutions. These technologies provided functionality and reliability comparable to proprietary systems, while offering greater flexibility and significantly reducing costs. The deployment highlighted the potential of open-source solutions to drive innovation in SG technologies without compromising on performance.

6.2.9 Deploy Modular and Scalable System Architecture Using Containerization

Various open-source software packages were containerized to create consistent and predictable software images. For some applications, pre-existing Docker files were unavailable, necessitating the creation of new Docker files to enable containerization. The Dockerization process demonstrated significant potential for simplifying the integration of diverse components, ensuring compatibility and ease of deployment.

6.3 Research Contributions

This study makes the following contributions to the field:

- Advancing SM Design: By combining existing technologies with the IEC 61850 standard, the research provides a comprehensive framework for modern smart metering systems that extend beyond traditional substation applications.
- Integration of IoT and Web Services: The thesis explored the use of IoT protocols and RESTful web services within an IEC 61850 environment, demonstrating new pathways for integrating diverse technologies into the SG.
- Standardization and Interoperability: The implementation highlights the importance of standardized communication protocols, reducing complexity and enabling seamless device integration in distributed energy networks.

6.4 Limitations and Challenges

Despite its successes, the study encountered several limitations:

- Security Considerations: While the proposed system provides a robust communication framework, additional measures are needed to address cybersecurity threats in distributed environments.
- Scalability of IoT Integration: Although the system supports IoT protocols, integrating a wider range of IoT devices and standards requires further exploration.
- Hardware Constraints: The performance of low-cost hardware, such as Raspberry Pi devices, may limit the system's applicability in large-scale deployments.
- Lack of Substation Integration: The research project was mainly computer driven mainly as access to substation hardware for integration was not possible. This constraint was due to the researcher not being in a single geographic location to make use of provided laboratory substation equipment.
- MMS Undeveloped open-source software: Making use of the MMS library was limiting as it was required to make use of a software library that was out of date. There were various manufacturers using their own version of the protocol. This makes it difficult for any student that does not have the funds for a brand license.

These limitations provide valuable opportunities for future research.

6.5 Future Work

Building on the foundation laid by this research, several areas merit further investigation:

6.5.1.1 Enhanced Security Mechanisms

Future systems should incorporate advanced cybersecurity frameworks, such as the IEC 62351 standard, to address vulnerabilities in communication and data exchange.

6.5.1.2 Expanded IoT Integration

Exploring additional IoT protocols, such as MQTT and CoAP, can enhance the system's versatility and make it compatible with a broader range of SG applications.

1. Real-World Implementation and Testing

2. Deploying the system in real-world scenarios would validate its performance and scalability, offering insights for further refinement and optimization.

3. Incorporating big data analytics and machine learning can improve energy management practices by providing actionable insights based on real-time and historical data.

4. Instead of using MMS to communicate outside the substation, investigate and implement the Routable GOOSE messaging.

6.6 Publications

Kruger R., Kriger C. (2024) ‘Cloud Computing approach to the IEC 61850 standard’, International Journal of Electrical and Electronic Engineering & Telecommunications (IJEEET) (submitted for publication)

6.7 Conclusion

This chapter summarized all the proposed deliverables and the extent to which they were achieved. The potential applications of this work in both industry and academia are discussed, highlighting its relevance and utilitization in real-world scenarios. A comprehensive list of the developed software and tools are provided to showcase the practical outcomes of this research. Additionally, possible avenues for future work are identified, offering insights into how this project could be further extended and improved. Finally, publications stemming from this research are listed, underscoring its contribution to the academic and professional discourse in the field.

REFERENCES

Ağin, A., Demirören, A. & Usta, Ö., 2024. A Novel Approach for Power System Protection Simulation via the IEC 61850 Protocol. *IEEE Access,* Volume 12, pp. 107656 - 107669.

Apostolov, A., 2017. R-GOOSE: what it is and its application in distribution automation. *CIRED,* 2017(1), pp. 1438-1441.

Burunkaya, M. & Pars, T., 2017 . *A Smart Meter Design and Implementation Using ZigBee Based Wireless Sensor Network in Smart Grid.* Ankara, IEEE.

Cavalieri, S., Cantali, G. & Susinna, A., 2022. Integration of IoT Technologies into the Smart Grid. *Sensors,* 2475(22), pp. 3-17.

Coles, R., 2021. *Web API basics – definition, technologies, features, and uses..* [Online]
Available at: https://www.gravitee.io/blog/web-apis-and-their-use#:~:text=A%20Web%20API%20of%20a,and%20access%20and%20authentication%20mechanisms.
[Accessed 08 04 2023].

Dalpiaz, G. et al., 2018. A battery-free non-intrusive power meterfor low-cost energy monitoring.

Das, N. et al., 2024. Exploring the Potential Application of IEC 61850 to Enable Energy Interconnectivity in Smart Grid Systems. *IEEE,* Volume 12, pp. 56910 - 56923.

Dewi, S. . D. T., Panatarani, C. & Joni, I. . M., 2016. Design and Development of DC High Current Sensor Using.

Dingra, S., 2016. *REST vs. SOAP: Choosing the best web service.* [Online] Available at: https://www.techtarget.com/searchapparchitecture/tip/REST-vs-SOAP-Choosing-the-best-web-service
[Accessed 04 08 2023].

docs, D., 2023. *Docker overview.* [Online] Available at: https://docs.docker.com/get-started/overview/
[Accessed 01 07 2023].

Gayo-Abeleira, M. et al., 2023. *Design and implementation of multiprotocol framework for residential prosumer incorporation in flexibility markets,* Madrid: Elsevier.

Hasam, A. M. & Kadhim, A. A., 2020. DESIGN AND IMPLEMENTATION OF SMART METER FOR SMART CITY. *IJICT,* 3(3), pp. 33-42.

Hau-feng, L. et al., 2020. *Application Research on the Substitution Specification of MMS.* Hangzhou, ICCEIC.

Haung, W., 2018. Learn IEC 61850 Configuration in 30 Minutes.

Hlaing, W., 2017. Implementation of WiFi-Based Single Phase Smart. *IEEE.*

Hou, S., Kong, F. & Liu, W., 2013. Research on IEC61850 Gateway base on MMS mapping. *Advanced Materials Research,* 694-697(1), pp. 2576-2580.

Hussain, S. S., Roomi, M. M., Mashima, D. & Chang, E.-C., 2023. *End-to-End Performance Evaluation of R-SV / R-GOOSE Messages for Wide Area Protection and Control Applications,* Singapore: IEEE.

Hussain, S. S., Ustun, T. S. & Kalam, A., 2019. A Review of IEC 62351 Security Mechanisms for IEC 61850 Message Exchanges. *IEE,* 16(9), pp. 5643 - 5654.

Hwang, S., 2021. Investigation of Wireless IEC 61850 MMS using Raspberry Pi. *Turkish Journal of Computer and Mathemetics Education,* 12(13), pp. 5770-5778.

IEC, 2003. *Part 7-2:Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI),* s.l.: IEC.

IEC, 2004. *Part 8-1: Specific Communication Service Mapping (SCSM)–Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3.* 2004-05 ed. s.l.:IEC.

Iglesias-Urkia, M. et al., 2018. Integrating Electrical Substations within the IoT using IEC 61850, CoAP and CBOR. *IEEE INTERNET OF THINGS JOURNAL,* 14(8), p. 6.

Iglesias-Urkia, M. et al., 2019. An Open-Source Hardware/Software IED based on IoT and IEC 61850 Standard. *IEEE Internet of things journal,* 6(5), pp. 7437-7449.

Iglesias-Urkia, M., Casado-Mansill, D., Mayer, S. & Urbieta, A., 2018. Validation of a CoAP to IEC 61850 Mapping and Benchmarking vs HTTP-REST and WS-SOAP. *IEEE,* 18(978-1-5386-7108-5), pp. 1015-1022.

Iglesias-Urkia, M., Urbieta, A., Parra, J. & Casado-Mansilla, D., 2017. *IEC 61850 meets CoAP: Towards the integration of Smart Grids and IoT standards,* Linz: IoT.

innovatorsgurus.com, 2023. *PZEM-004T V3.0 User Manual.* [Online] Available at: https://innovatorsguru.com/wp-content/uploads/2019/06/PZEM-004T-V3.0-Datasheet-User-Manual.pdf
[Accessed 2023 07 05].

Jasim, G. M. & Abdalla, K. K., 2020. Single Phase Energy Smart Meter System Design and. *ICEST.*

Karnouskos, S., da Silv, P. . G. & Ili´c, . D., 2013. Developing a Web Application for Monitoring and Management of Smart Grid Neighborhoods. *IEEE,* 13(978-1-4799-0752-6), p. 408.

Kriger, C., Behardien, S. & Retonda-Modiya, J.-C., 2013. A Detailed Analysis of the GOOSE Message Structure in an IEC. *INT J COMPUT COMMUN, ISSN 1841-9836.*

Luc, J., 2023. *modbus-tk.* [Online] Available at: https://github.com/ljean/modbus-tk
[Accessed 2023 07 05].

Midul, A. H. S. et al., 2023. *Design and Implementation of IoT-Based Smart Energy Meter to Augment Residential Energy Consumption.* Bangladhesh, IEE.

Murray, M., 2023. *control high voltage devices with an arduino uno.* [Online] Available at: https://www.thegeekpub.com/234976/control-high-voltage-devices-with-an-arduino/
[Accessed 2 07 2023].

nn-digital-website, 2023. *example of pzem 004t.* [Online] Available at: https://www.nn-digital.com/en/blog/2019/11/04/example-of-the-pzem-004t-v3-v3-0-interfacing-program-using-arduino/
[Accessed 02 07 2023].

Oh, S.-Y., Lee, Y.-J. & Hwang, S.-H., 2023. *Design and Implementation of IoT Gateway with MQTT and IEC 61850 MMS Protocol,* s.l.: s.n.

Park, J. et al., 2012 . *IEC 61850 Standard Based MMS Communication Stack Design Using OOP.* Seoul, South Korea, IEEE.

Pedersen, A. B., Hauksson, E. B. & Andersen, P. B., 2010. Facilitating a Generic Communication. *Facilitating a Generic Communication Interface to Distributed Energy Resources Mapping IEC 61850 to RESTful Services,* 10(IEE), pp. 61-65.

Pham, G. T., 2013. *Integration of IEC61850 MMS and LTE to support smart metering communication,* Twente: University of twente.

Pi, R., 2023. *Raspberry Pi OS.* [Online] Available at: https://www.raspberrypi.com/software/ [Accessed 06 06 2023].

Pramudhita, A. N., Asmara, R. A., Siradjuddin, I. & Rohadi, E., 2018. *Internet of Things Integration in Smart Grid.* s.l., IEEE.

Raspberry Pi, 2023. *raspberry-pi-4.* [Online] Available at: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html [Accessed 05 07 2023].

Ruland, C., Kang, N. & Sassmannshausen, J., 2016. *Rejuvination of the IEC61850 protocol stack for MMS.* Seoul, IEEE.

Rusnak, O., 2022. *https://cedalo.com/blog/mqtt-connection-beginners-guide/.* [Online] Available at: Olga Rusnak [Accessed 10 09 2023].

Sayed, S., Hussain, T., Gastli, A. & Benammar, M., 2019. Design and realization of an open-source and modular smart meter. *Energy Science & engineering,* 7(4), pp. 1405-1422.

SISCO, 1995. *Overview and Introduction to the Manufacturing Message Specification.* 2 ed. Sterling Heights(Michigan): SISCO.

Ustun, T. S. & Hussain, S. M. S., 2017. EC 62351-4 Security Implementations for IEC 61850 MMS Messages. *IEEE Access,* Volume 20, pp. 1-9.

Wang, D. et al., 2008. Research on Distributed Transmission of Power Telecontrol Information Based on ACSI/MMS. *IEEE,* 978-1-4244-1718-6(08), pp. 670-674.

Yun, Z., Junjie, L., Ji, C. & Hua, W., 2017. A Framework Research of Power Distribution. *IEEE,* p. 5.

APPENDICES

Python Code interacting with PZEM400T module:

```
import requests
import time
import json
import serial
import modbus_tk.defines as cst
from modbus_tk import modbus_rtu

TOKEN_FILE = "token.txt" #output file name where smart-meter token wil be kept

def get_token(): # This declares information about the meter to be submitted to the POST endpoint.
    data = {
    "location":"Claremont",
    "name":"k-Residence"
    }

    response = requests.post(token_url,json=data)
    if response.status_code == 200:
        print(response)
        return response.json().get('id')
    else:
        print(f"Failed to retrieve token from {url}.")
        return None

def save_token(token):
    with open(TOKEN_FILE, "w") as file: #Open the file in write mode
        file.write(token)

def load_token():
    try:
        with open(TOKEN_FILE, "r") as file: #Open the file in read mode
            return file.read().strip()
```

```
    except FileNotFoundError:
        return None

def send_post_request(url, data):  #creating a new meter via the API
    response = requests.post(url, json=data)
    print(f"POST request sent to {url}. Response: {response.status_code}")

# Set the token endpoint URL
token_url = "http://{{ URL }}/SmartMeter"

# Set the API endpoint URL
api_url = "http://{{ URL }}/Readings/"

# Set the interval between requests in seconds
interval = 60

try:
    # Connect to the usb via serial
   #This is where we interface with the PZEM-400T device
    serial_port = serial.Serial(
        port='/dev/ttyUSB0',
        baudrate=9600,
        bytesize=8,
        parity='N',
        stopbits=1,
        xonxoff=0
    )

    master = modbus_rtu.RtuMaster(serial_port)
    master.set_timeout(2.0)
    master.set_verbose(True)
    dict_payload = dict()

    # Retrieve or load the token – This registers a new meter if there is no token in the
token.txt file
    token = load_token()
```

```
    if not token:
        token = get_token()
        if token:
            save_token(token)

    if token:

        while True:
            # First we create a json payload of all the readings gathered
            data = master.execute(1, cst.READ_INPUT_REGISTERS, 0, 10)
            print(data)
            dict_payload["voltage"] = data[0] / 10.0
            dict_payload["current"] = (data[1] + (data[2] << 16)) / 1000.0  # [A]
            dict_payload["power"] = (data[3] + (data[4] << 16)) / 10.0  # [W]
            dict_payload["powerFactor"] = data[8] / 100.0

            str_payload = json.dumps(dict_payload, indent=2)
            print(str_payload)

            send_post_request(api_url+""+token, dict_payload) # Here we post the reading
to the readings endpoint using the token as identification



            time.sleep(interval) # Just a delay to create a rest time between uploads

    else:
        print("Token retrieval failed. Exiting...")

except KeyboardInterrupt:
    print('Exiting script due to keyboard interrupt.')
except Exception as e:
    print(f'An error occurred: {e}')
finally:
    if 'master' in locals():
        master.close()
```

Template for Smart Meter SCD File

```
<?xml version="1.0" encoding="UTF-8"?>

<SCL revision="B" version="2007" xmlns="http://www.iec.ch/61850/2003/SCL" schemaLocation="http://www.iec.ch/61850/2003/SCL SCL.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <Header id="SCL_Header" version="SCL_Header" revision="0" toolID="smart-meter-api">

   <History>

    <Hitem version="1" revision="0" when="2023/07/23 09:17:35" who="Ryan Kruger" what="Ryan Kruger" />

   </History>

 </Header>

 <Substation>

   <name>virtual smart meter substation</name>

   <desc>virtual smart meter substation</desc>

 </Substation>

 <IED name="SM-0e29ab93-08ad-4992-9e11-147a33c043f6" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">

   <AccessPoint name="AP">

    <Server>

     <LDevice inst="CTRL">

       <LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />

       <LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />

       <LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />

       <LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
```

```
        <LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
      </LDevice>
    </Server>
  </AccessPoint>
  <Services nameLength="64">
    <DynAssociation max="10" />
    <ConfLogControl max="10" />
    <GOOSE max="10" />
    <GetDirectory />
    <GetDataObjectDefinition />
    <DataObjectDirectory />
    <GetDataSetValue />
    <SetDataSetValue />
    <DataSetDirectory />
    <ConfDataSet modify="true" maxAttributes="50" max="50" />
    <DynDataSet max="100" maxAttributes="50" />
    <ReadWrite />
    <ConfReportControl bufConf="true" bufMode="both" max="50" />
    <GetCBValues />
    <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
    <LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />
    <GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />
    <FileHandling />
```

```
    <ConfLNs fixPrefix="true" />
    <ConfLdName />
    <ConfSigRef max="100" />
   </Services>
  </IED>
  <IED  name="SM-bce841af-25ca-4688-856c-96e7416b1214"  manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">
   <AccessPoint name="AP">
    <Server>
     <LDevice inst="CTRL">
      <LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />
      <LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />
      <LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />
      <LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
      <LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
     </LDevice>
    </Server>
   </AccessPoint>
   <Services nameLength="64">
    <DynAssociation max="10" />
    <ConfLogControl max="10" />
    <GOOSE max="10" />
    <GetDirectory />
    <GetDataObjectDefinition />
```

<DataObjectDirectory />

<GetDataSetValue />

<SetDataSetValue />

<DataSetDirectory />

<ConfDataSet modify="true" maxAttributes="50" max="50" />

<DynDataSet max="100" maxAttributes="50" />

<ReadWrite />

<ConfReportControl bufConf="true" bufMode="both" max="50" />

<GetCBValues />

<ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn" cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />

<LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />

<GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />

<FileHandling />

<ConfLNs fixPrefix="true" />

<ConfLdName />

<ConfSigRef max="100" />

</Services>

</IED>

<IED name="SM-457968f3-48fd-4e01-ba3d-568fa8a76f95" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">

<AccessPoint name="AP">

<Server>

<LDevice inst="CTRL">

```
      <LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />
      <LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />
      <LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />
      <LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
      <LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
    </LDevice>
  </Server>
</AccessPoint>
<Services nameLength="64">
  <DynAssociation max="10" />
  <ConfLogControl max="10" />
  <GOOSE max="10" />
  <GetDirectory />
  <GetDataObjectDefinition />
  <DataObjectDirectory />
  <GetDataSetValue />
  <SetDataSetValue />
  <DataSetDirectory />
  <ConfDataSet modify="true" maxAttributes="50" max="50" />
  <DynDataSet max="100" maxAttributes="50" />
  <ReadWrite />
  <ConfReportControl bufConf="true" bufMode="both" max="50" />
  <GetCBValues />
```

<ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn" cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />

<LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />

<GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />

<FileHandling />

<ConfLNs fixPrefix="true" />

<ConfLdName />

<ConfSigRef max="100" />

</Services>

</IED>

<IED name="SM-9b0a5a48-0cd8-484c-b4ca-f86e44ea47b1" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">

<AccessPoint name="AP">

<Server>

<LDevice inst="CTRL">

<LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />

<LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />

<LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />

<LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />

<LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />

</LDevice>

</Server>

</AccessPoint>

<Services nameLength="64">

```
      <DynAssociation max="10" />
      <ConfLogControl max="10" />
      <GOOSE max="10" />
      <GetDirectory />
      <GetDataObjectDefinition />
      <DataObjectDirectory />
      <GetDataSetValue />
      <SetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet modify="true" maxAttributes="50" max="50" />
      <DynDataSet max="100" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl bufConf="true" bufMode="both" max="50" />
      <GetCBValues />
      <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
      <LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />
      <GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />
      <FileHandling />
      <ConfLNs fixPrefix="true" />
      <ConfLdName />
      <ConfSigRef max="100" />
    </Services>
  </IED>
```

<IED name="SM-6ae81dda-0de5-40ca-8c16-6d82dfa6335f" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">
<AccessPoint name="AP">
<Server>
<LDevice inst="CTRL">
<LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />
<LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />
<LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />
<LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
<LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
</LDevice>
</Server>
</AccessPoint>
<Services nameLength="64">
<DynAssociation max="10" />
<ConfLogControl max="10" />
<GOOSE max="10" />
<GetDirectory />
<GetDataObjectDefinition />
<DataObjectDirectory />
<GetDataSetValue />
<SetDataSetValue />
<DataSetDirectory />
<ConfDataSet modify="true" maxAttributes="50" max="50" />

<DynDataSet max="100" maxAttributes="50" />

<ReadWrite />

<ConfReportControl bufConf="true" bufMode="both" max="50" />

<GetCBValues />

<ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn" cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />

<LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />

<GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />

<FileHandling />

<ConfLNs fixPrefix="true" />

<ConfLdName />

<ConfSigRef max="100" />

</Services>

</IED>

<IED name="SM-cbfe4619-88da-4574-a20b-1caa2d501589" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">

<AccessPoint name="AP">

<Server>

<LDevice inst="CTRL">

<LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />

<LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />

<LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />

<LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />

<LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />

```
      </LDevice>
    </Server>
  </AccessPoint>
  <Services nameLength="64">
    <DynAssociation max="10" />
    <ConfLogControl max="10" />
    <GOOSE max="10" />
    <GetDirectory />
    <GetDataObjectDefinition />
    <DataObjectDirectory />
    <GetDataSetValue />
    <SetDataSetValue />
    <DataSetDirectory />
    <ConfDataSet modify="true" maxAttributes="50" max="50" />
    <DynDataSet max="100" maxAttributes="50" />
    <ReadWrite />
    <ConfReportControl bufConf="true" bufMode="both" max="50" />
    <GetCBValues />
    <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
    <LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />
    <GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />
    <FileHandling />
    <ConfLNs fixPrefix="true" />
```

```
    <ConfLdName />
    <ConfSigRef max="100" />
   </Services>
  </IED>
  <IED  name="SM-d50b20fd-0aaf-40cc-8764-c618468de595"  manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">
   <AccessPoint name="AP">
    <Server>
     <LDevice inst="CTRL">
      <LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />
      <LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />
      <LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />
      <LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
      <LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
     </LDevice>
    </Server>
   </AccessPoint>
   <Services nameLength="64">
    <DynAssociation max="10" />
    <ConfLogControl max="10" />
    <GOOSE max="10" />
    <GetDirectory />
    <GetDataObjectDefinition />
    <DataObjectDirectory />
```

```
      <GetDataSetValue />
      <SetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet modify="true" maxAttributes="50" max="50" />
      <DynDataSet max="100" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl bufConf="true" bufMode="both" max="50" />
      <GetCBValues />
      <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
      <LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />
      <GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />
      <FileHandling />
      <ConfLNs fixPrefix="true" />
      <ConfLdName />
      <ConfSigRef max="100" />
    </Services>
  </IED>
  <IED name="SM-f44f550a-3b08-482d-aa24-6a585286261e" manufacturer="smart-
meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">
    <AccessPoint name="AP">
      <Server>
        <LDevice inst="CTRL">
          <LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />
```

```
        <LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />
        <LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />
        <LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />
        <LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />
      </LDevice>
    </Server>
  </AccessPoint>
  <Services nameLength="64">
    <DynAssociation max="10" />
    <ConfLogControl max="10" />
    <GOOSE max="10" />
    <GetDirectory />
    <GetDataObjectDefinition />
    <DataObjectDirectory />
    <GetDataSetValue />
    <SetDataSetValue />
    <DataSetDirectory />
    <ConfDataSet modify="true" maxAttributes="50" max="50" />
    <DynDataSet max="100" maxAttributes="50" />
    <ReadWrite />
    <ConfReportControl bufConf="true" bufMode="both" max="50" />
    <GetCBValues />
    <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
```

<LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />

<GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />

<FileHandling />

<ConfLNs fixPrefix="true" />

<ConfLdName />

<ConfSigRef max="100" />

</Services>

</IED>

<IED name="SM-a87afd3a-5238-470c-84c4-8762b9b13545" manufacturer="smart-meter-api" configVersion="1.0" originalSclRevision="B" originalSclVersion="2007">

<AccessPoint name="AP">

<Server>

<LDevice inst="CTRL">

<LN lnType="LLN0_2007" lnClass="LLN0" inst="" prefix="" />

<LN lnType="LPHD_TYPE" lnClass="LPHD" inst="1" prefix="" />

<LN lnType="MMXN_TYPE" lnClass="MMXN" inst="1" prefix="" />

<LN lnType="CSWI_TYPE" lnClass="CSWI" inst="1" prefix="" />

<LN lnType="XSWI_TYPE" lnClass="XSWI" inst="1" prefix="" />

</LDevice>

</Server>

</AccessPoint>

<Services nameLength="64">

<DynAssociation max="10" />

<ConfLogControl max="10" />

```
      <GOOSE max="10" />
      <GetDirectory />
      <GetDataObjectDefinition />
      <DataObjectDirectory />
      <GetDataSetValue />
      <SetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet modify="true" maxAttributes="50" max="50" />
      <DynDataSet max="100" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl bufConf="true" bufMode="both" max="50" />
      <GetCBValues />
      <ReportSettings rptID="Dyn" trgOps="Dyn" intgPd="Dyn" optFields="Dyn"
cbName="Conf" datSet="Dyn" bufTime="Dyn" resvTms="true" owner="true" />
      <LogSettings trgOps="Dyn" intgPd="Dyn" datSet="Dyn" logEna="Dyn" />
      <GSESettings appID="Dyn" dataLabel="Dyn" datSet="Dyn" cbName="Conf" />
      <FileHandling />
      <ConfLNs fixPrefix="true" />
      <ConfLdName />
      <ConfSigRef max="100" />
    </Services>
  </IED>
  <DataTypeTemplates>
    <LNodeType id="LLN0_2007" lnClass="LLN0">
```

<DO name="NamPlt" type="LPL_LD2007" />

<DO name="Beh" type="ENS_Beh" />

<DO name="Health" type="ENS_Health" />

<DO name="Mod" type="ENC_Mod_direct" />

<DO name="LocKey" type="SPS_noSVnoBL" />

<DO name="Loc" type="SPS_noSVnoBL" />

<DO name="LocSta" type="SPC_direct" />

<DO name="Diag" type="SPC_direct" />

<DO name="LEDRs" type="SPC_direct" />

<DO name="MltLev" type="SPG_SP" />

</LNodeType>

<LNodeType id="LPHD_TYPE" lnClass="LPHD">

<DO name="PhyNam" type="DPL_Full" />

<DO name="PhyHealth" type="ENS_Health" />

<DO name="OutOv" type="SPS_noSVnoBL" />

<DO name="Proxy" type="SPS_noSVnoBL" />

<DO name="InOv" type="SPS_noSVnoBL" />

<DO name="OpTmh" type="INS_noSVnoBL" />

<DO name="NumPwrUp" type="INS_noSVnoBL" />

<DO name="WrmStr" type="INS_noSVnoBL" />

<DO name="WacTrg" type="INS_noSVnoBL" />

<DO name="PwrUp" type="SPS_noSVnoBL" />

<DO name="PwrDn" type="SPS_noSVnoBL" />

<DO name="PwrSupAlm" type="SPS_noSVnoBL" />

```
  <DO name="RsStat" type="SPC_direct" />
  <DO name="Sim" type="SPC_direct" />
</LNodeType>
<LNodeType id="MMXN_TYPE" lnClass="MMXN">
  <DO name="Beh" type="ENS_BehaviourModeKind" />
  <DO name="Amp" type="MV" />
  <DO name="Vol" type="MV" />
  <DO name="Watt" type="MV" />
  <DO name="PwrFact" type="MV" />
  <DO name="Hz" type="MV" />
</LNodeType>
<LNodeType id="CSWI_TYPE" lnClass="CSWI">
  <DO name="Beh" type="TMW_Generated_ENS_BehaviourModeKind" />
  <DO name="Pos" type="DPC" />
</LNodeType>
<LNodeType id="XSWI_TYPE" lnClass="XSWI">
  <DO name="Beh" type="TMW_Generated_ENS_BehaviourModeKind" />
  <DO name="Loc" type="SPS" />
  <DO name="OpCnt" type="INS" />
  <DO name="SwTyp" type="ENS_SwitchFunctionKind" />
  <DO name="Pos" type="DPC" />
  <DO name="BlkOpn" type="SPC" />
  <DO name="BlkCls" type="SPC" />
</LNodeType>
```

```
<DAType id="AnalogueValue">
  <BDA name="f" bType="FLOAT32" />
</DAType>
<DOType id="DPL_Full" cdc="DPL">
  <DA name="vendor" bType="VisString255" fc="DC">
    <Val>TMW</Val>
  </DA>
  <DA name="hwRev" bType="VisString255" fc="DC" />
  <DA name="swRev" bType="VisString255" fc="DC" />
  <DA name="serNum" bType="VisString255" fc="DC" />
  <DA name="model" bType="VisString255" fc="DC" />
  <DA name="location" bType="VisString255" fc="DC" />
  <DA name="name" bType="VisString64" fc="DC" />
  <DA name="owner" bType="VisString255" fc="DC" />
  <DA name="ePSName" bType="VisString255" fc="DC" />
  <DA name="primeOper" bType="VisString255" fc="DC" />
  <DA name="secondOper" bType="VisString255" fc="DC" />
  <DA name="latitude" bType="FLOAT32" fc="DC" />
  <DA name="longitude" bType="FLOAT32" fc="DC" />
  <DA name="altitude" bType="FLOAT32" fc="DC" />
  <DA name="mRID" bType="VisString255" fc="DC" />
  <DA name="d" bType="VisString255" fc="DC" />
  <DA name="dU" bType="Unicode255" fc="DC" />
</DOType>
```

```
  <DOType id="DPC" cdc="DPC">
    <DA name="stVal" bType="Dbpos" fc="ST" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
    <DA name="ctlModel" bType="Enum" fc="CF" dchg="true" type="CtlModelKind" valKind="RO">
      <Val>status-only</Val>
    </DA>
  </DOType>
  <DOType id="ENC_Mod_direct" cdc="ENC">
    <DA name="origin" bType="Struct" fc="ST" type="Originator" />
    <DA name="stVal" bType="Enum" fc="ST" type="BehaviourModeKind" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
    <DA name="ctlModel" bType="Enum" fc="CF" type="CtlModelKind" valKind="RO">
      <Val>direct-with-normal-security</Val>
    </DA>
    <DA name="d" bType="VisString255" fc="DC" />
    <DA name="dU" bType="Unicode255" fc="DC" />
    <DA name="Oper" bType="Struct" fc="CO" type="OperBehaviourModeKind" />
  </DOType>
  <DOType id="ENS_Beh" cdc="ENS">
```

<DA name="stVal" bType="Enum" fc="ST" type="BehaviourModeKind" dchg="true" dupd="true" />

<DA name="q" bType="Quality" fc="ST" qchg="true" />

<DA name="t" bType="Timestamp" fc="ST" />

<DA name="d" bType="VisString255" fc="DC" />

<DA name="dU" bType="Unicode255" fc="DC" />

</DOType>

<DOType id="ENS_Health" cdc="ENS">

<DA name="stVal" bType="Enum" type="HealthKind" fc="ST" dchg="true" dupd="true" />

<DA name="q" bType="Quality" fc="ST" qchg="true" />

<DA name="t" bType="Timestamp" fc="ST" />

<DA name="d" bType="VisString255" fc="DC" />

<DA name="dU" bType="Unicode255" fc="DC" />

</DOType>

<DOType id="ENS_BehaviourModeKind" cdc="ENS">

<DA name="stVal" bType="Enum" type="BehaviourModeKind" fc="ST" dchg="true" dupd="true" />

<DA name="q" bType="Quality" fc="ST" qcgh="true" />

<DA name="t" bType="Timestamp" fc="ST" />

</DOType>

<DOType id="ENS_SwitchFunctionKind" cdc="ENS">

<DA name="stVal" bType="Enum" fc="ST" dchg="true" type="SwitchFunctionKind" />

<DA name="q" bType="Quality" fc="ST" qchg="true" />

```
  <DA name="t" bType="Timestamp" fc="ST" />
</DOType>
<DOType id="INS" cdc="INS">
  <DA name="stVal" bType="INT32" fc="ST" dchg="true" dupd="true" />
  <DA name="q" bType="Quality" fc="ST" qchg="true" />
  <DA name="t" bType="Timestamp" fc="ST" />
</DOType>
<DOType id="INS_noSVnoBL" cdc="INS">
  <DA name="stVal" bType="INT32" fc="ST" dchg="true" />
  <DA name="q" bType="Quality" fc="ST" qchg="true" />
  <DA name="t" bType="Timestamp" fc="ST" />
  <DA name="d" bType="VisString255" fc="DC" />
  <DA name="dU" bType="Unicode255" fc="DC" />
</DOType>
<DOType id="LPL_LD2007" cdc="LPL">
  <DA name="vendor" bType="VisString255" fc="DC">
    <Val>TMW</Val>
  </DA>
  <DA name="swRev" bType="VisString255" fc="DC">
    <Val>1.0</Val>
  </DA>
  <DA name="d" bType="VisString255" fc="DC" />
  <DA name="dU" bType="Unicode255" fc="DC" />
  <DA name="configRev" bType="VisString255" fc="DC">
```

<Val>1.0</Val>

</DA>

<DA name="paramRev" bType="INT32" fc="ST" dchg="true">

<Val>0</Val>

</DA>

<DA name="valRev" bType="INT32" fc="ST" dchg="true">

<Val>0</Val>

</DA>

<DA name="ldNs" bType="VisString255" fc="EX">

<Val>IEC 61850-7-4:2007</Val>

</DA>

</DOType>

<DOType id="MV" cdc="MV">

<DA name="mag" bType="Struct" type="AnalogueValue" fc="MX" dchg="true" dupd="true" />

<DA name="q" bType="Quality" type="" fc="MX" qchg="true" />

<DA name="t" bType="Timestamp" type="" fc="MX" />

</DOType>

<DAType id="OperBehaviourModeKind">

<BDA name="ctlVal" bType="Enum" type="BehaviourModeKind" />

<BDA name="origin" bType="Struct" type="Originator" />

<BDA name="ctlNum" bType="INT8U" />

<BDA name="T" bType="Timestamp" />

<BDA name="Test" bType="BOOLEAN" />

```
    <BDA name="Check" bType="Check" />
  </DAType>
  <DAType id="OperBool">
    <BDA name="ctlVal" bType="BOOLEAN" />
    <BDA name="origin" bType="Struct" type="Originator" />
    <BDA name="ctlNum" bType="INT8U" />
    <BDA name="T" bType="Timestamp" />
    <BDA name="Test" bType="BOOLEAN" />
    <BDA name="Check" bType="Check" />
  </DAType>
  <DAType id="Originator">
    <BDA name="orCat" bType="Enum" type="OriginatorCategoryKind" />
    <BDA name="orIdent" bType="Octet64" />
  </DAType>
  <DOType id="SPC" cdc="SPC">
    <DA name="ctlModel" bType="Enum" type="CtlModelKind" fc="CF" dchg="true"
valKind="RO">
      <Val>status-only</Val>
    </DA>
  </DOType>
  <DOType id="SPC_direct" cdc="SPC">
    <DA name="origin" bType="Struct" fc="ST" type="Originator" />
    <DA name="stVal" bType="BOOLEAN" fc="ST" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
```

```
    <DA name="t" bType="Timestamp" fc="ST" />
    <DA name="ctlModel" bType="Enum" fc="CF" type="CtlModelKind" valKind="RO">
      <Val>direct-with-normal-security</Val>
    </DA>
    <DA name="d" bType="VisString255" fc="DC" />
    <DA name="dU" bType="Unicode255" fc="DC" />
    <DA name="Oper" bType="Struct" fc="CO" type="OperBool" />
  </DOType>
  <DOType id="SPG_SP" cdc="SPG">
    <DA name="setVal" bType="BOOLEAN" fc="SP" dchg="true" />
    <DA name="d" bType="VisString255" fc="DC" />
    <DA name="dU" bType="Unicode255" fc="DC" />
  </DOType>
  <DOType id="SPS" cdc="SPS">
    <DA name="stVal" bType="BOOLEAN" fc="ST" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
  </DOType>
  <DOType id="SPS_noSVnoBL" cdc="SPS">
    <DA name="stVal" bType="INT32" fc="ST" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
    <DA name="d" bType="VisString255" fc="DC" />
```

```
    <DA name="dU" bType="Unicode255" fc="DC" />
  </DOType>
  <DOType id="TMW_Generated_DPC" cdc="DPC">
    <DA name="stVal" bType="Dbpos" fc="ST" dchg="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
    <DA name="ctlModel" bType="Enum" type="CtlModelKind" fc="CF" dchg="true" valKind="RO">
      <Val>status-only</Val>
    </DA>
  </DOType>
  <DOType id="TMW_Generated_ENS_BehaviourModeKind" cdc="ENS">
    <DA name="stVal" bType="Enum" fc="ST" type="BehaviourModeKind" dchg="true" dupd="true" />
    <DA name="q" bType="Quality" fc="ST" qchg="true" />
    <DA name="t" bType="Timestamp" fc="ST" />
  </DOType>
  <EnumType id="BehaviourModeKind">
    <EnumVal ord="1">on</EnumVal>
    <EnumVal ord="2">blocked</EnumVal>
    <EnumVal ord="3">test</EnumVal>
    <EnumVal ord="4">test/blocked</EnumVal>
    <EnumVal ord="5">off</EnumVal>
  </EnumType>
```

```
<EnumType id="CtlModelKind">
  <EnumVal ord="0">status-only</EnumVal>
  <EnumVal ord="1">direct-with-normal-security</EnumVal>
  <EnumVal ord="2">sbo-with-normal-security</EnumVal>
  <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
  <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
</EnumType>
<EnumType id="HealthKind">
  <EnumVal ord="1">Ok</EnumVal>
  <EnumVal ord="2">Warning</EnumVal>
  <EnumVal ord="3">Alarm</EnumVal>
</EnumType>
<EnumType id="OriginatorCategoryKind">
  <EnumVal ord="0">not-supported</EnumVal>
  <EnumVal ord="1">bay-control</EnumVal>
  <EnumVal ord="2">station-control</EnumVal>
  <EnumVal ord="3">remote-control</EnumVal>
  <EnumVal ord="4">automatic-bay</EnumVal>
  <EnumVal ord="5">automatic-station</EnumVal>
  <EnumVal ord="6">automatic-remote</EnumVal>
  <EnumVal ord="7">maintenance</EnumVal>
  <EnumVal ord="8">process</EnumVal>
</EnumType>
<EnumType id="SwitchFunctionKind">
```

```
      <EnumVal ord="1">Load Break</EnumVal>
      <EnumVal ord="2">Disconnector</EnumVal>
      <EnumVal ord="3">Earthing Switch</EnumVal>
      <EnumVal ord="4">High Speed Earthing Switch</EnumVal>
    </EnumType>
  </DataTypeTemplates>
</SCL>
```