

*Eirik Hammer*  
*Eilev Sivertsen*

# **Analysis and implementation of the IEC 61850 standard**

Master's Thesis, Spring 2008



*Eirik Hammer*  
*Eilev Sivertsen*

# **Analysis and implementation of the IEC 61850 standard**

Master's Thesis, Spring 2008

M. Sc. Thesis  
**Analysis and Design of the 61850 Standard**

March, 2008

Written by:

s001740, Eilev Sivertsen \_\_\_\_\_

s011688, Eirik Hammer \_\_\_\_\_



## **Abstract**

In many areas of engineering, interoperability is a goal when technical systems are designed. This is true also for the domain of electrical engineering and in particular for substation automation systems. The IEC61850 standard addresses this challenge and is the focus of this thesis. The thesis analyses the IEC61850 standard and gives an overview of its content. The analysis places the Brodersen RTU32 in relation to the scope of the standard. A basic IEC61850 server is designed and implemented for the RTU which runs under Windows CE. The system consists of an information model and an information exchange model and is capable of basic client/server communication. Basic services, such as reporting and logging are implemented which allow a client such as a SCADA system to review historical data for a substation and receive reports based on events in the substation. An SCL parser is included in the implementation which allows a substation to be configured according to the SCL configuration file format defined in the IEC61850 standard.

## Preface

This report is the documentation of a final thesis submitted for the degree Master of Science in Engineering at the Technical University of Denmark, DTU. The thesis has been carried out in cooperation with the Department of Informatics and Mathematical Modeling (IMM), Centre for Electric Technology (CET) and Brodersen Controls A/S.

Bjarne Poulsen at IMM, Chresten Træholt at CET and Ole Borgbjerg from Brodersen Controls A/S have been supervisors for the thesis.

Recommended prerequisites for reading the report are a basic knowledge of software engineering and substation automation.

The thesis has been carried out in the period September 3, 2007 to March 14, 2008.

*Eirik Hammer*  
*Eilev Sivertsen*

*Kongens Lyngby, 2008*

## Acknowledgements

First, we would like to thank our supervisor Bjarne Poulsen at IMM for great guidance, inspiration and patience during the whole project. Second, we would like to thank Chresten Træholt at CET for feedback and advice. We would also like to thank Ole Borgbjerg and Beggi Oskarsson at Brodersen Controls A/S for their cooperation, feedback and availability. We also thank Brodersen Controls A/S for making a Brodersen RTU32 available for the whole duration of this project. Finally, we would like to thank Preben Nyeng for valuable feedback and advice.



# CONTENTS

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.1.1	About Brodersen Controls A/S . . . . .	1
1.1.2	IEC61850 . . . . .	2
1.1.3	Brodersen RTU32 . . . . .	3
1.2	Motivation . . . . .	4
1.2.1	Motivation for the Authors . . . . .	4
1.2.2	Motivation for the Company . . . . .	5
1.2.3	Motivation for the Supervisors . . . . .	5
1.3	Vision . . . . .	5
1.4	Problem Statement . . . . .	6
1.5	Development Method . . . . .	7
1.6	About the report . . . . .	8
<b>2</b>	<b>Analysis</b>	<b>9</b>
2.1	Analysis of the IEC61850 standard . . . . .	9
2.1.1	Basic Concepts of IEC61850 . . . . .	9
2.1.2	The IEC61850 Standard - Overview and Scope . . . . .	11
2.1.3	Data Model . . . . .	15
2.1.4	Substation Configuration Description Language . . . . .	16
2.1.5	Abstract Communication Service Interface . . . . .	18
2.1.6	Information Models . . . . .	20
2.1.7	Information Exchange . . . . .	23
2.1.8	Communication . . . . .	28
2.2	Analysis of Scenarios . . . . .	30
2.2.1	Scenario 1: Event based single alarm . . . . .	30
2.2.2	Scenario 2: Event based double alarm . . . . .	33
2.2.3	Scenario 3: Control physical output pulse on RTU . . . . .	34
2.2.4	Scenario 4: Send alarm setpoint to RTU . . . . .	35
2.2.5	Scenario 5: Event report of analogue input . . . . .	35
2.2.6	Conclusion to Analysis of scenarios . . . . .	36
2.3	Analysis of the Brodersen RTU32 . . . . .	36
2.4	Specification of Requirements . . . . .	37
2.5	Conclusion . . . . .	38

<b>3</b>	<b>Design</b>	<b>40</b>
3.1	Architecture of Solution . . . . .	40
3.2	Information Model . . . . .	41
3.3	Information Exchange Model . . . . .	44
3.3.1	Unbuffered Reporting . . . . .	46
3.3.2	Buffered Reporting . . . . .	46
3.3.3	Logging . . . . .	47
3.3.4	Observer Pattern . . . . .	48
3.4	Communication . . . . .	49
3.5	Device Model . . . . .	51
3.6	Substation Module . . . . .	52
3.7	SCL Configuration . . . . .	54
3.8	Conclusion . . . . .	54
<b>4</b>	<b>Implementation</b>	<b>57</b>
4.1	Implementation of IEC61850 System . . . . .	57
4.2	Information Model . . . . .	59
4.3	Information Exchange Model . . . . .	62
4.4	Communication Module . . . . .	64
4.5	Device Module . . . . .	65
4.6	Substation Module . . . . .	66
4.7	Conclusion . . . . .	68
<b>5</b>	<b>Test</b>	<b>69</b>
5.1	Unit Test . . . . .	69
5.2	Test Cases . . . . .	69
5.2.1	GetDataDirectory . . . . .	70
5.2.2	GetDataSetDirectory . . . . .	70
5.2.3	Logging . . . . .	70
5.2.4	Reporting . . . . .	70
5.2.5	Connecting to the Server . . . . .	71
5.3	Conclusion . . . . .	71
<b>6</b>	<b>Conclusion</b>	<b>72</b>
6.1	Summary of Results . . . . .	72
6.2	Summary of Contributions . . . . .	74
6.3	Discussion and Future Work . . . . .	74
6.3.1	Improvements to the Basic IEC61850 Server Implementation . .	75
6.3.2	Additional IEC61850 Functionality . . . . .	75
<b>A</b>	<b>Glossary</b>	<b>79</b>
<b>B</b>	<b>Details of the IEC61850 Standard</b>	<b>82</b>
B.1	List of Logical Node Groups . . . . .	82
B.2	ACSI Classes and Their Services . . . . .	82



# LIST OF FIGURES

---

1.1	A Brodersen RTU32 . . . . .	4
1.2	Time Plan for Development Process . . . . .	8
2.1	Conceptual modeling approach of the IEC61850 standard . . . . .	10
2.2	IEC61850 communication profile placed in an RTU setup . . . . .	13
2.3	IEC61850 communication profile placed in a LAN setup . . . . .	14
2.4	The hierarchy of the IEC61850 data model . . . . .	15
2.5	Information model and information exchange model of ACSI . . . . .	18
2.6	The SCSMs of IEC61850 placed according to the OSI layers [8] . . . . .	29
2.7	Sequence Diagram for Association . . . . .	32
2.8	Sequence Diagram for GetServerDirectory . . . . .	33
2.9	Sequence Diagram for SetURCBValues . . . . .	33
3.1	Architecture of solution . . . . .	41
3.2	UML Class Diagram for the information model module with attributes, properties and methods . . . . .	42
3.3	Elaborated sequence diagram for services GetServerDirectory and Get-LogicalDeviceDirectory . . . . .	44
3.4	UML Class Diagram for the information exchange model module with attributes, properties and methods . . . . .	45
3.5	Elaborated sequence diagram for reporting prior to server/client communication . . . . .	47
3.6	Elaborated sequence diagram for buffered reporting from server to client . . . . .	47
3.7	Elaborated sequence diagram for logging . . . . .	48
3.8	UML Class Diagram for the general Observer design pattern . . . . .	49
3.9	Methods of the communication module . . . . .	50
3.10	UML Class Diagram for the Device module with attributes, properties and methods . . . . .	51
3.11	Sequence Diagram for update of data model based on changes in inputs . . . . .	52
3.12	Sequence Diagram for update of RTU based on changes in outputs . . . . .	52
3.13	UML Class Diagram for the Substation module with attributes, properties and methods . . . . .	53
3.14	UML Class Diagram for the whole system . . . . .	56

# LIST OF TABLES

---

2.1.1	The file types of SCL as defined in [7]. All file types are in XML format, but they contain different elements depending on their purpose. . . . .	17
2.1.2	Example of types: The logical node MMXU. The recursive structure of types DATA and data attributes is illustrated. . . . .	22
B.1.1	List of Logical Node Groups . . . . .	83
B.2.1	Complete List of ACSI Classes and Their Services . . . . .	84
B.2.2	. . . . .	84

# INTRODUCTION

---

In this chapter the background and motivation for focusing this project on the IEC61850 standard will be outlined. The vision for the project will be stated and also the structure of the report is briefly explained.

## 1.1 Background

In this section a brief introduction will be given to Brodersen Controls A/S, the IEC61850 standard and the Brodersen RTU32.

### 1.1.1 About Brodersen Controls A/S

Brodersen Controls A/S was founded back in 1970 when it was known as Brodersen Teknik A/S. During the nearly 40 years since its founding, Brodersen Controls A/S has developed into one of Europe's leading designers and manufacturers of process IT components. The head office is located in Denmark and there are also subsidiaries in the United Kingdom, Germany and the United States.

Brodersen Controls A/S offer a wide range of products: RTU<sup>1</sup> modules for telemetry<sup>2</sup> and data logging, communications modules for GSM, GPRS and Radio modems and Fieldbus modules. They also offer products such as power supplies, timers, operator panels and relays. These products and other solutions from Brodersen Controls A/S are used in industries such as:

- Water distribution and wastewater
- Oil and gas
- Electric power distribution

---

<sup>1</sup>Remote Terminal Unit - explained in 1.1.3

<sup>2</sup>A technology that allows remote measurement and reporting of information of interest to the system designer or operator

- Transportation (airport, railways, traffic control)
- Telecommunication

Brodersen Controls A/S became ISO 9001 certified in march 2005 [3].

### 1.1.2 IEC61850

Substation automation is essential in order to maintain an efficient and reliable electrical infrastructure. The IEC61850 standard is developed to make this automation interoperable and cost-efficient. The IEC61850 standard has a number of benefits compared to previous standards which are often referred to as legacy standards. These can be described as *'artifacts of the eighthies'* - the time in which many of them were developed. The communication protocols of these legacy standards were developed for serial link technology and were later adapted to run over TCP/IP-Ethernet [18]. From the start, one of the objectives of the legacy protocols was to account for bandwidth limitations by minimizing the number of bytes sent. Many of these protocols were proprietary and thus communication between devices from different vendors was generally not possible.

From the start, the IEC61850 standard was designed to operate over modern networking technologies. Interoperability is ensured by the standard and many features are included which it would be impossible to include using previous standards.

Compared to legacy standards, a few of the specific benefits of the IEC61850 standard include the following features:

- Every element of data is named using descriptive strings whereas legacy protocols often use storage location and register numbers to identify data
- The communication protocol supports GOOSE<sup>3</sup>, GSSE<sup>4</sup>, SMV<sup>5</sup> and many other services not supported in legacy protocols
- The standard includes a standardized configuration language for substations, SCL<sup>6</sup>, which uses XML<sup>7</sup> files for the configuration of a device and removes ambiguity issues in previous standards

---

<sup>3</sup>Generic Object Oriented Substation Event - an abstract data model mapping in the communications protocol

<sup>4</sup>Generic Substation Status Event

<sup>5</sup>Sampled Measured Values

<sup>6</sup>Substation Configuration Language

<sup>7</sup>Extensible Markup Language - Widely used markup language which facilitates sharing of structured data across platforms, typically over the Internet

- The use of GOOSE and GSSE over LAN<sup>8</sup> removes the need for wiring separate links for each relay, thus lowering installation costs
- The use of a single merging unit supporting SMV lowers transducer and maintenance costs
- Less manual configuration is needed for client applications and devices, reducing errors and lowering commissioning cost [18]

### 1.1.3 Brodersen RTU32

RTU is short for Remote Terminal Unit. An RTU is a microprocessor controlled electronic device used in e.g. SCADA<sup>9</sup> systems for collecting telemetry data and transmitting them to a central server. The RTU itself is installed at the remote location as the name indicates. An RTU can also receive commands from the central server and issue these to the remote system. RTUs are generally equipped with input channels for sensing or metering, output channels for control, indication or alarms and a communication port.

The Brodersen RTU32 is a new and advanced RTU which combines the functionality and performance of telemetry RTUs, PLCs<sup>10</sup> and industrial PCs. It is based on a 32-bit 300MHz CPU and runs the Windows CE 5.0<sup>11</sup> operating system and thus it is flexible with regard to installing industrial and utility applications. It also has the Straton development tool which contains a virtual machine and an IDE<sup>12</sup>. This enables the user of the RTU to write his or her own applications in the five languages supported by the IEC61131-3 standard<sup>13</sup>, and test and run these applications on the RTU.

In order to make the RTU32 compatible with PLCs and flow computers as well as communication networks, it has been designed to support several protocols such as SNMP<sup>14</sup>, TCP/IP<sup>15</sup> and the IEC60870-5-101/104 utility protocol. SNMP, for instance, is used by network management systems and allows monitoring of devices attached to a network. Network and SNMP settings, as well as other general settings on the

---

<sup>8</sup>Local Area Network

<sup>9</sup>Supervisory Control And Data Acquisition

<sup>10</sup>Programmable Logic Controller - a small special-purpose computer used to automate machines

<sup>11</sup>Windows CE is a variation of Windows designed for minimalistic computers and embedded systems

<sup>12</sup>Integrated Developers Environment

<sup>13</sup>Standard for PLCs. This particular part deals with programming languages of PLCs and the five languages are: Ladder diagram, Function block diagram, Structured text, Instruction list and Sequential function chart

<sup>14</sup>Simple Network Management Protocol

<sup>15</sup>Transmission Control Protocol/Internet Protocol



RTU32, can be configured via the Ethernet network interface either locally or remotely by using a web browser[4].

For input and output, the RTU32 is equipped with 16 digital inputs, 4 relay outputs, 4 analogue inputs and 2 analogue outputs. Furthermore it can be easily extended with extension modules which can have various combinations of extra input and output channels. An RTU32 can be seen in figure 1.1.



Figure 1.1: A Brodersen RTU32

## 1.2 Motivation

There are several reasons why the IEC61850 standard was chosen as the subject for this master thesis project. In this section some of the motivating factors for the authors, for the collaborating company Brodersen and for the supervisors will be outlined.

### 1.2.1 Motivation for the Authors

Software development may be exciting and interesting by itself, but generally its value does not really materialise before it performs some useful, practical task which would otherwise be infeasible to have performed. Therefore, a software engineer is often going to work closely together with experts in other fields, who have an interest in obtaining software that makes their task easier. For such a piece of software to become efficient, it is sometimes required that the software engineer have a thorough

understanding of the domain in which the software is to be used. By choosing this project, the authors have also chosen to work with the domain of electrical engineering and the IEC61850 in particular. Since there is a great deal of new development in this area, it is likely that working with this project could give a professional knowledge of the field of electrical engineering which could benefit a future career. Also, the complexity of the domain will be a challenge during the project.

Most students at DTU plan to get a job at some company after finishing the studies. During the studies one often becomes used to solving certain academical tasks, which have been designed to teach the students some particular piece of theory. When working in a company outside the four quadrants of DTU, tasks are often very different from what a student is used to. Collaborating with Brodersen on a project like this will likely serve to bridge the gap between studies and a professional career.

### **1.2.2 Motivation for the Company**

Since Brodersen Controls A/S is engaged in manufacturing products and solutions used in electric power distribution, such as RTU modules for telemetry and data logging, it is a natural step to incorporate a standard like IEC61850 into their range of products. Hereby they can demonstrate that their RTUs are capable of fulfilling the role of a central component in the IEC61850's new communication structure which experts agree will be the most widely used in this field in the future.

### **1.2.3 Motivation for the Supervisors**

DTU is involved in a large research project, called NextGen [5], concerning dynamic markets in networks of virtual power plants. The goal of the project is to integrate the electricity system with information systems allowing electricity to be produced and controlled in a decentralised manner compared to today. The vision is that this is how electricity will be produced for the coming generation. The supervisors for this project are also part of this larger research project. In virtual power plant networks and the NextGen project, the IEC61850 standard is essential, since it prescribes how the communication in such networks takes place. The IEC61850 standard has been pointed out as the basis of the communication profile in the NextGen project.

## **1.3 Vision**

For this project the vision of the authors is to give an overview of the IEC61850 standard and to implement a suitable part of an IEC61850 server on a Brodersen RTU32. The task consists of two quite different parts, where first a thorough analysis of the

standard must be made so that a helpful overview can be presented to Brodersen. Second, a server must be designed and implemented on the RTU which complies with the standard and can make basic IEC61850 server functionality available to a SCADA system.

The vision of Brodersen Controls A/S is that the project might contribute to the development of one of the company's strategic hardware platforms used as a component for supervision and control of power transmission networks and substation automation.

In more detail, Brodersen want the project to help them

- gain insight into how the IEC61850 standard can be practically applied and interpreted
- place the RTU32 as a component in the new communication structure implied in the IEC61850 standard
- implement a basic IEC61850 server on the RTU32 Windows CE platform with well-defined interfaces for
  - configuration of a server driver (such as API<sup>16</sup>, file etc)
  - read/write physical and virtual/internal input/output in RTU32 low level database WTOOLS32.dll (in Win32)
- test an IEC61850 server in RTU32 against a ZenOn SCADA Client for evaluating performance

A number of cases have been provided by Brodersen Controls A/S to illustrate scenarios for which the final product might be used. These scenarios will be presented in section 2.2. Along with the scenarios, the vision of the authors and the vision and detailed wishes of Brodersen Controls A/S will be used as an inspiration to prepare a problem statement in the following.

## 1.4 Problem Statement

This project has two separate dimensions in that it consists of an analysis and an implementation of the IEC61850 standard. The analysis of the standard is necessary both in order to provide the desired overview of its content and scope, and also with regard to beginning to make an implementation of the standard. The overall task can be described as easing Brodersen's employment of the IEC61850 standard on their RTU32 by providing a useful overview of the contents of the standard and constructing a generic implementation of a suitable part of the standard that can be

---

<sup>16</sup>Application Programming Interface

easily extended or modified according to the needs and wishes of the company.

Because of the comprehensive nature of the IEC61850 standard, a suitable demarcation is necessary to pinpoint the focus area of the software implementation of this project. This will be given in the specification of requirements in section 2.4, following a thorough analysis of the standard in chapter 2. The part of the standard that is chosen and implemented, shall be chosen based on the analysis of the standard, the vision for the project and the analysis of the scenarios and wishes of Brodersen Controls A/S. The overall problem statement can thus be formulated as follows:

### **Problem Statement**

- An analysis shall be performed which provides a general overview of the IEC61850 standard in terms of functionality and scope. The possibility of employing a Brodersen RTU32 in this communication structure shall be kept in mind.
- The scenarios provided by Brodersen Controls A/S shall be analysed based on the IEC61850 standard.
- A suitable part of the IEC61850 standard shall be designed and implemented to run on the Brodersen RTU32 under Windows CE.

## **1.5 Development Method**

In order for a relatively large software project like this to be completed successfully, it is useful to follow a development process. For this project, a modified waterfall model has been chosen. According to the experience of the authors, the waterfall model is used for most smaller study-related projects, but the shortcomings of the model become more and more apparent the larger the projects are. To counteract these shortcomings, students often modify the model - often not explicitly - to contain overlapping phases. "The waterfall model with overlapping phases" is also called the Sashimi model and this is the approach that has been chosen for this project.

This model neutralises some of the major drawbacks of the unmodified waterfall model in which one phase must be completed before the next can be commenced. When developing a project according to the unmodified waterfall model, it is hence impossible to gain knowledge from for instance the testing phase and use this knowledge to modify the design or requirements phase. With the Sashimi model such insight can be used to modify previous stages and this is an advantage in terms of having the solution constructed on schedule and developed according to a well-defined specification of requirements.

The time plan for the development process is shown in figure 1.2. As the figure shows, the domain analysis has taken a significant amount of time. This stems from the fact that the IEC61850 standard is new to the authors and is a large domain to become sufficiently acquainted with to be able to implement.



Figure 1.2: Time Plan for Development Process

## 1.6 About the report

In chapter 2 an analysis of the IEC61850 standard is given. Chapter 3 describes the design phase of the project. The implementation is described in chapter 4. The tests of the implementation are described in chapter 5. Finally, chapter 6 concludes the report.

It has been aimed to make the report as easily readable as possible. However, it is assumed that the reader has some basic prerequisites with regard to software development. Given that the software shall be developed for employment in the domain of electrical engineering, some basic knowledge of this field is an advantage, but a software engineering student should be able to get a reasonable grasp of the subject by reading the report.

In appendix A an explanation is given of the terms which a software engineering student cannot be expected to be familiar with. Some terms are also explained in footnotes the first time they are encountered. Appendix B contains details of the IEC61850 standard which the report makes references to.

Enclosed with the report, a CD should be found which contains all the source code for the IEC61850 server.

# ANALYSIS

---

The goal of this chapter is hence to give an overview of the contents of the IEC61850 standard and then to delimit the scope of the project based on the analysis of the scenarios and the RTU.

Section 2.1 will present the analysis of the IEC61850 standard where first, a general introduction is given to the contents of the different parts of the standard. This should provide the reader with an overview and basic understanding of the standard. A more thorough analysis then follows which explains parts of the standard in more detail, which should give insight into the ideas and methods of the standard. In section 2.2 the scenarios provided by Brodersen Controls A/S are analysed with the goal of detailing which parts of the standard are required in order to implement functionality corresponding to the scenarios. Section 2.3 presents a short analysis of the Brodersen RTU32. The specification of requirements is given in section 2.4, based on the analysis. To conclude the chapter, a summary of the analysis is presented in 2.5.

## 2.1 Analysis of the IEC61850 standard

The purpose of this section is to provide insight and overview of how the standard is structured and how it works. First, basic concepts of the standard are explained and then a brief overview is given of the contents of the standard. Afterwards, the different parts of the standard are inspected individually and analysed in more detail. Later in the chapter this will be the basis for a delimitation of what the software of this project shall comprise.

### 2.1.1 Basic Concepts of IEC61850

Some basic concepts will be briefly introduced before the details in the standard are explained.

A substation can be defined as a node in an electrical power network where lines and cables are connected for transmission and distribution of electric power [2]. A substation often has the capability of transforming electricity, usually from high to low voltage for distribution by a low-voltage network. Most substations therefore have one or more transformers and they may have many other functions as well, such as switching, breaking and protection capabilities. A substation automation system (SAS) is a computer system which allows e.g. an administrator to communicate with the substation over a computer network such as the internet. Obviously, when developing such a system it is necessary to create a model of a general substation with all of its components and functions. Then it is necessary to stipulate the exact form of communication that is allowed and supported by the system. This describes exactly the challenges addressed by the IEC61850 standard.

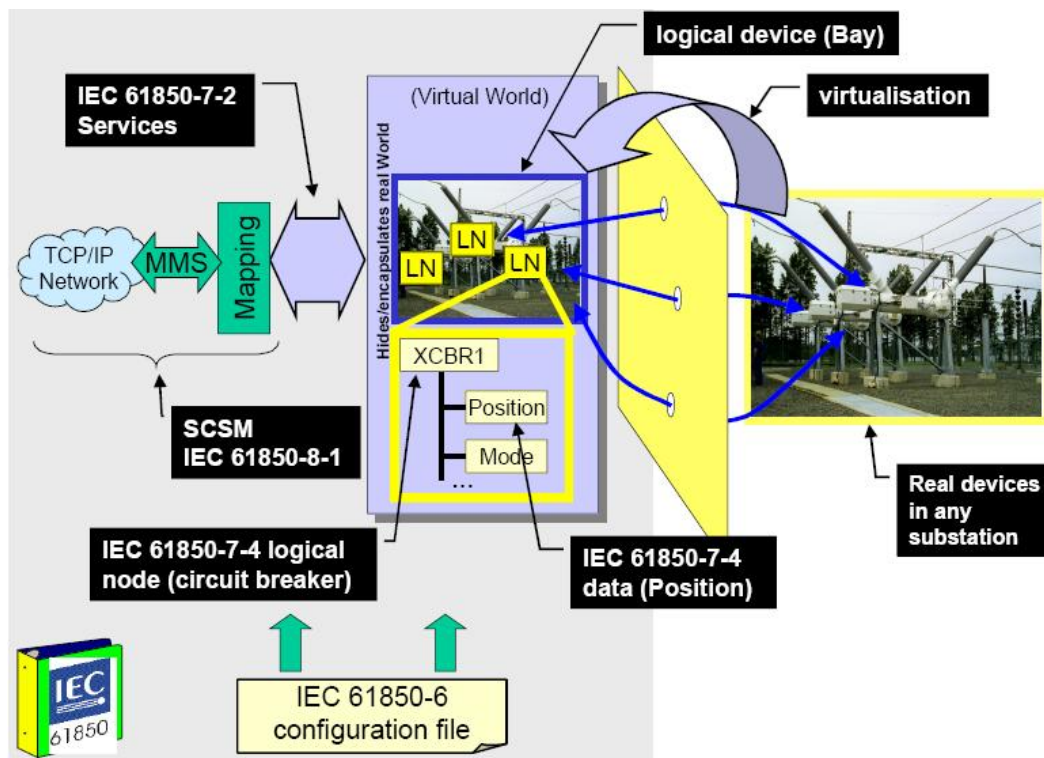


Figure 2.1: Conceptual modeling approach of the IEC61850 standard. A real physical substation is modelled into a virtual substation. The virtual substation contains a detailed data model encapsulating the real world objects and services are mapped to a network communication protocol. Relevant parts of IEC61850 are shown. Figure occurs in [9].

Figure 2.1 illustrates how the substation is virtualised into a data model suitable for a computer system. This data model consists of a number of logical nodes, which are the key objects in the model of the IEC61850 standard. A logical node can have a number of data objects attached to it, and each data object can have a number of data

attributes. The data model is explained more thoroughly in section 2.1.3.

A substation can often comprise a number of IEDs. When an IED is added, the extension must be reflected in the particular instance of the data model modelling the substation. This instance should be extended correspondingly. As is also illustrated in figure 2.1, the IEC61850 standard allows for configuration and modifications to a SAS, through the use of SCL which is defined in IEC61850-6 ([7]). This language is explained in section 2.1.4.

An IEC61850 server provides a number of services for a client. For instance, logging, reporting and settings control is possible. All the services are defined in part 7-2 of the standard named Abstract Communication Service Interface (ACSI). Many of the services of the ACSI are explained in section 2.1.5 of this analysis.

A client that is able to connect to a substation is sometimes referred to as a SCADA system or a control center.

The data model and ACSI define the structure and form of the content communicated by the IEC61850 server to clients. The ACSI can be mapped to a specific communication service mapping (SCSM). While other SCSMs could be used as far as the ACSI is concerned, specific communication mappings are defined in IEC61850-8 and IEC61850-9. These are explained in section 2.1.8.

## 2.1.2 The IEC61850 Standard - Overview and Scope

The general title of the IEC61850 standard is *Communication networks and systems in substations*. The standard consists of the following parts:

- **IEC61850-1** Introduction and overview
- **IEC61850-2** Glossary  
Explains terms and abbreviations used throughout the standard
- **IEC61850-3** General requirements  
Specifies system requirements with emphasis on the quality requirements of the communication network.
- **IEC61850-4** System and project management  
Specifies system and project management with respect to the engineering process, life cycle of overall system and IEDs<sup>1</sup>, and the quality assurance.

---

<sup>1</sup>Intelligent Electronic Devices



- **IEC61850-5** Communication requirements for function and device models  
Describes all required functions in order to identify communication requirements between technical services and the substation, and between IEDs within the substation. The goal is interoperability for all interactions.
- **IEC61850-6** Substation automation system configuration description language  
Specifies the SCL file format for describing communication related IED configurations, IED parameters, communication system configurations, function structures, and the relations between them. The purpose is to exchange IED capability description, and SA<sup>2</sup> system descriptions between IED engineering tools and different system engineering tools.
- **IEC61850-7** Basic communication structure for substation and feeder equipment
  - **IEC61850-7-1** Principles and models  
Introduces modelling methods, communication principles and information models used in IEC61850-7. Also, detailed requirements and explanations are given regarding the relation between IEC61850-7-x and the requirements from IEC51850-5.
  - **IEC61850-7-2** Abstract communication service interface (ACSI)  
Presents the ACSI providing abstract interfaces describing the communications between a client and a remote server, such as interfaces for data access and retrieval, device control, event reporting and logging.
  - **IEC61850-7-3** Common data classes  
Specifies common attribute types and common data classes related to substation applications. The common data classes specified, are for instance, classes for status information, measured information, controllable status information, controllable analogue set point information, status settings and analogue settings.
  - **IEC61850-7-4** Compatible logical node classes and data classes  
Specifies the compatible logical node names and data names for communication between IEDs.
- **IEC61850-8** Specific communication service mapping (SCSM)
  - **IEC61850-8-1** Mapping to MMS<sup>3</sup> (ISO/IEC 9506 Part 1 and Part 2)  
Specifies how time-critical and non-time-critical data may be exchanged through local area networks by mapping ACSI to MMS.
- **IEC61850-9** Specific communication service mapping (SCSM)
  - **IEC61850-9-1** Serial unidirectional multidrop point to point link  
Specifies the specific communication service mappings for the communica-

---

<sup>2</sup>Substation Automation

<sup>3</sup>Manufacturing Message Specification - An international networking standard

tion between bay and process level and a mapping of the abstract service for the transmission of sampled values. These are specified on a serial unidirectional multidrop point to point link.

– **IEC61850-9-2** Mapping on a IEEE 802.3 based process

Defines the SCSM for the transmission of sampled values according to the abstract specification in IEC61850-7-2.

• **IEC61850-10** Conformance testing

Specifies how a SAS<sup>4</sup> should be tested to ensure conformance with the IEC61850 standard.

In order to get an outline of where exactly the IEC61850 communication standard and an RTU appear in relation to substation automation systems, two typical setups shall be presented. These two scenarios are illustrated in figures 2.2 and 2.3. Other setups are also conceivable but these two figures underline the difference between a traditional setup with an RTU and the intended LAN setup of the future. These two setups shall be referred to as the RTU setup and the LAN setup.

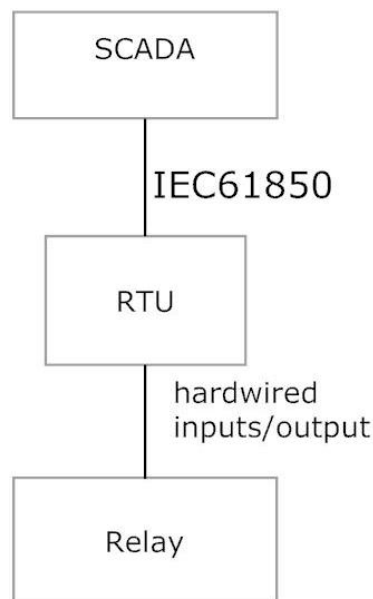


Figure 2.2: IEC61850 communication profile placed in an RTU setup. The RTU is hardwired to other substation equipment and IEC61850 is the means of communication between RTU and SCADA system. Inspired by figures in [16] and [8]

In the RTU setup (figure 2.2, devices are typically hardwired to the input and output ports of the RTU. In this case, the IEC61850 standard can allow a traditional substation setup to comply with the new standard on the communication side towards the

<sup>4</sup>Substation Automation System

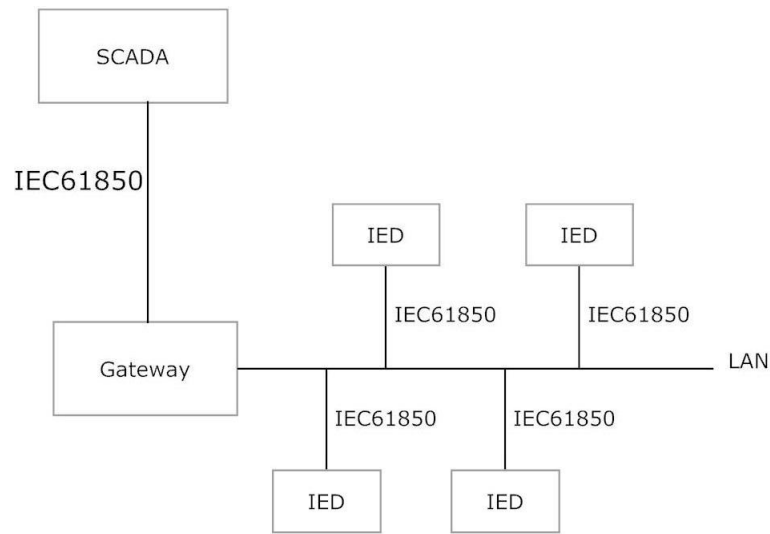


Figure 2.3: IEC61850 communication profile placed in a LAN setup. IEC61850 is the means of communication between IEDs and gateway (over LAN) and also between SCADA and gateway (over TCP/IP or LAN). Inspired by figures in [16] and [8]

control center. This might be advantageous if an IEC61850 compliant SCADA system is to be connected to a traditional RTU setup.

In the LAN setup, the IEC61850 is used both in the communication between IEDs in the SAS and between the SAS and the SCADA system in which case it is usually between the gateway and SCADA. However, the IEDs may also be able to communicate with the SCADA. Since the standard has been developed also for this kind of communication, it is clear that this is the type of setup envisioned as the typical setup for the future. However, it is likely that the RTU setup will be around in the substation automation area for many years to come, since it is a domain where it usually takes a long time for existing technology to be completely replaced by new technology.

Figure 2.1 explains the conceptual modelling approach of the IEC61850 standard. It also shows which parts of the standard deal with certain aspects of the substation automation system and the communication involved.

Parts 6 to 9-2 constitute the main parts of the standard. Therefore these parts will be the focus of the analysis in this chapter. These parts define the way a substation is modelled with data classes and services. The SCL language is defined, which allows a user to configure a substation automation system or a single IED connected to the system. Also, the abstract communication service interface (ACSI) is defined which lists all the services available for a client. Finally, it is explained how these abstract services could be mapped to a specific communication service mapping (SCSM).

### 2.1.3 Data Model

As was illustrated in figure 2.1, logical nodes are key objects in the IEC61850 data model. The data model is hierarchical and logical nodes are the essential elements of this model. A logical node represents a particular function within a device and can be defined as “the smallest part of a function that exchanges data” [6](3.5). The IEC61850 standard defines 91 different logical node classes which are grouped together into 13 logical node groups according to their functionality (for a complete list, refer to Appendix B.1). These are defined in [9] and each LN is defined as a class with certain attributes.

In an instance of the data model, some of the logical node instances may be grouped together into a bay which is defined as closely connected subparts of the substation with some common functionality [6](3.10). A bay is thus a logical grouping, not necessarily a physical device. In the hierarchical data model, it can be represented by a logical device.

The hierarchy of the data model is illustrated in figure 2.4. In a substation there can be one or more physical devices. A physical device has one or more servers and a server is the topmost object in the hierarchical data model. A logical device is a more fine-grained grouping of functionality related to a particular physical device. The logical device is contained in a server. Thus, one server may have more than one logical device and a logical device may contain several logical nodes.

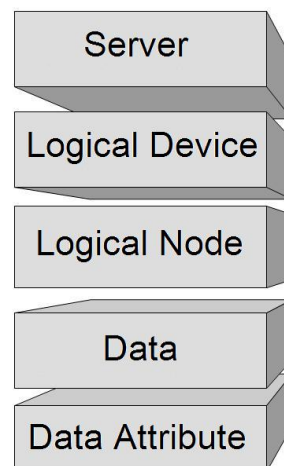


Figure 2.4: The hierarchy of the data model of IEC61850. One server may contain more than one logical device (LD) and so on. 91 different LNs are defined which inherit from one abstract LN class. Each LN contains certain mandatory and optional DATA.

Further down in the hierarchical model, a logical node has a number of data objects

attached to it, and each of the data objects have a number of attributes with values.

Part of what makes logical nodes the pivotal point of the data model is the fact that they are predefined. Whereas the logical device and server may be decided upon individually by manufacturers or administrators of a substation, there are predefined logical nodes which cover all the necessary functionality of substation components. This is a benefit considering the goal of the standard which is to achieve interoperability between devices from different vendors.

Hubert Kirrmann of ABB Research Center states that: “Although IEC 61850 is defined as a ‘communication structure for substation and feeder equipment’ its main contribution is the definition of an object model for all substation objects” [15]. It is clear that since the standard has interoperability as a goal, its data model is of essential importance, and therefore it is an advantage that all functions can be modelled precisely and by predefined objects.

An important aspect of the object model is the fact that users are allowed to name substation components in a meaningful way. This is a consequence of the object oriented approach used for developing the standard.

The standard defines an object reference to differentiate between a reference to an object and the object name. The object reference is important in terms of implementation and is based on the data model in a straightforward manner. The object reference is comprised of the objects ordered hierarchically according to the data model and with dots between them. The general format is:

`LD/LN.Data.DataAttribute`

but this will be slightly extended later.

## **2.1.4 Substation Configuration Description Language**

A substation may be altered in structure for instance if one or more IEDs are added. Such additions can be defined by use of an SCL file. The SCL language allows for configuration of a substation both before employment but also as further equipment is added to the substation. SCL is short for Substation automation system Configuration description Language and it is defined in [7].

The SCL file format is used for describing communication related IED configurations, IED parameters, communication system configurations, function structures, and the relations between them. The purpose is to exchange IED capability description, and

Extension	Name	Description
.icd	IED Capability Description	Defines complete capability of an IED. Contains single IED description, optional communication system description and optional substation description
.ssd	System Specification Description	Complete specification of SAS excluding IED descriptions
.scd	Substation Configuration Description	Complete specification of SAS including IED descriptions
.cid	Configured IED Description	Makes communication possible between an IED and a IED configuration tool.

Table 2.1.1: The file types of SCL as defined in [7]. All file types are in XML format, but they contain different elements depending on their purpose.

substation automation system descriptions between IED engineering tools and different system engineering tools.

The SCL language is made up of four file types, each with a specific purpose. The types are shown in table 2.1.1. Any SCL file is structured with XML format and is made up of some of the following five parts, depending upon its purpose:

1. Header
2. Substation description
3. IED description
4. Communication system description
5. Data type templates

Table 2.1.1 shows in which file types the different parts occur. In order to be fully compliant with the IEC61850 standard, an IED shall have an ICD file containing basic information about the IED such as which logical nodes and services it supports, the IP address and so on. A configuration tool can then read such files and generate or modify an SCD file which describes the full substation configuration. This file should be based not only on the ICD files, but also on information entered by a system integrator about the substation prior to adding the IEDs, or alternatively an SSD file describing the SAS itself [7][1].

An SCD file used in this project is shown in Appendix C.

## 2.1.5 Abstract Communication Service Interface

Part 7-2 of IEC61850 [9] is dedicated entirely to the Abstract Communication Service Interface, abbreviated ACSI. The ACSI provides a number of abstract interfaces and it can be said that the ACSI represents the full capability of an IEC61850 Server as seen from a client. Some of the interfaces describe communications between a client and a remote server while other interfaces are provided for communication between an application in one device and remote applications in other devices. This could be system-wide event distribution or transmission of sampled measured values. The communications between a client and a remote server could be device control, reporting of events, logging of events, publisher/subscriber and others.

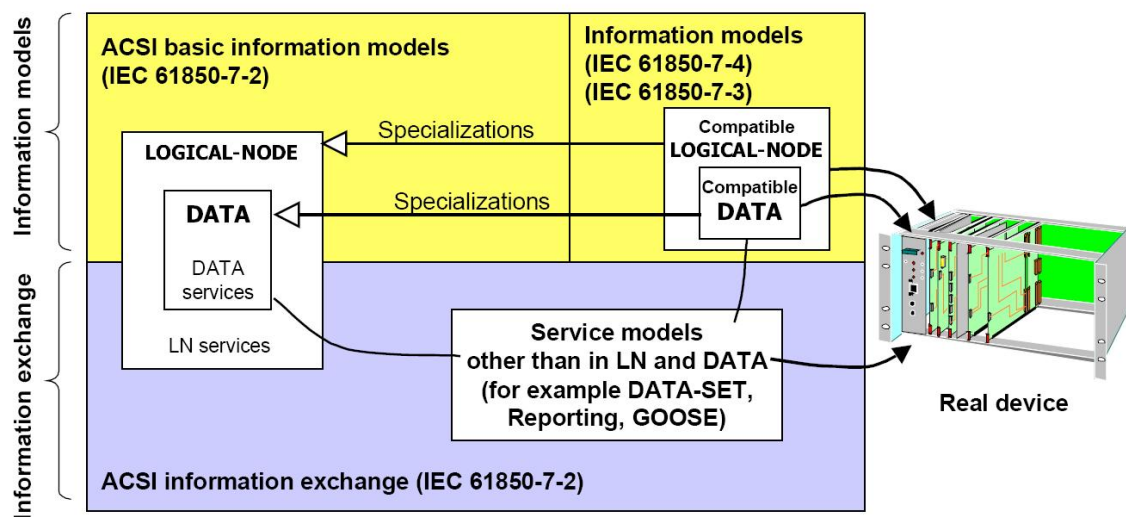


Figure 2.5: Conceptual model of ACSI which is comprised of both the information model (basic service models) and the information exchange model (other service models) [9]

The model of the ACSI, described in [9], defines an information model and an information exchange model. The information model corresponds with the models defined in [10] and [11] (“Common data classes” and “Compatible logical node classes and data classes”) and the relationship between the information model and the information exchange model is illustrated in figure 2.5, showing an excerpt of the conceptual model of the ACSI.

As can be seen from figure 2.5 the model consists of, firstly, ACSI basic information models which can be conceived as specialisations of the information models defined in IEC61850-7-3 and 7-4 ([10] and [11]) along with basic service models. Secondly, it consists of an information exchange model involving service models other than Logi-

cal Node and Data classes. An outline of these two models is presented below.

### **Information Model**

- **SERVER** - represents the external visible behaviour of a device. All other ACSI models are part of the server
- **Application association** - allows a client to be associated (connected) with a server
- **LOGICAL-DEVICE (LD)** - contains the information produced and consumed by a group of domain-specific application functions. Functions are defined as Logical Nodes.
- **LOGICAL-NODE (LN)** - contains the information produced and consumed by a domain-specific application function, for example, overvoltage protection or circuit-breaker.
- **DATA** - provide means to specify typed information, for example, position of a switch with quality information and timestamp, contained in LNs.

### **Information Exchange Model**

- **DATA-SET** - permits the grouping of data and data attributes.
- **Substitution** - supports replacement of a process value by another value
- **SETTING-GROUP-CONTROL-BLOCK** - defines how to switch from one set of setting values to another one and how to edit setting groups
- **REPORT-CONTROL-BLOCK** and **LOG-CONTROL-BLOCK** - describe the conditions for generating reports and logs based on parameters set by the client
- **Control blocks for generic substation events (GSE)** - supports a fast and reliable system-wide distribution of input and output data values.
- **Control blocks for transmission of sampled values** - fast and cyclic transfer of samples, for example, of instrument transformers
- **Control** - describes the services to control, for example, devices
- **Time and time synchronization** - provides the time base for the device and system
- **File transfer** - defines the exchange of large data blocks such as programs

Each of the items listed above correspond to one ACSI class and these are defined with class syntax, class attributes and types throughout [9]. The complete list of ACSI



classes and their services is shown in table B.2.1. Details as to how responses and error messages shall be given, are also given in this part of the standard. To give an overview and understanding of these classes and services, most of them are explained briefly in the analysis which follows. The order shown above shall be used and the models shall be divided into information models and information exchange models. The analysis will leave out the majority of the details provided in the standard, but might still be perceived as rather detailed and difficult to read.

## 2.1.6 Information Models

### SERVER

The SERVER class (defined in clause 6 of [9]) represents the externally visible behaviour of a device. The class contains an attribute `ServiceAccessPoint` which identifies the server within a system. It is an abstraction of an address to identify the server in the underlying SCSM. The class also contains a list of Logical Devices where at least one Logical Device must be contained. Furthermore it contains the attributes `File`, `TPAppAssociation` and `MCAAppAssociation`. These are (possibly empty) lists containing files on the server, two-party application associations and multicast application associations (explained in next section). The SERVER class provides the service `GetServerDirectory` which a client can use to retrieve the definition of the accessible information of a server. The service returns the LDs contained in the server.

### Application Associations

Application associations are defined in clause 7 of [9] and the association model defines two association classes, TPAA and MCAA, for two-party and multicast application associations, as well as access control concepts. The role of these associations is to make sure that reports and logs are transmitted to the correct clients. The class TPAA provides a bi-directional connection-oriented information exchange where services are confirmed. The attributes are `AssociationId` specifying an identification for the association and `AuthenticationParameter` containing user identification, view (access rights) and password. There are three services defined, `Associate`, `Abort` and `Release`. The class MCAA provides a unidirectional information exchange where services are unconfirmed. The server acts as a publisher and transmits information to one or more subscribers (clients). MCAA is used by a number of services which transmit data to all clients according to subscriptions.

## LOGICAL-DEVICE

The LOGICAL-DEVICE (abbreviated LD) is a composition of LOGICAL-NODE (abbreviated LN). It is defined in clause 8 of [9] and can be used simply as a container of a group of logical nodes. The attributes of the class are the name of the instance, path name of the instance and a list of contained logical nodes. There is one service provided, namely `GetLogicalDeviceDirectory`, which provides a client with a list of all contained logical nodes.

## LOGICAL-NODE

The LOGICAL-NODE class (abbreviated LN) is a composition of a number of other classes, including DATA, DATA-SET, BRCB, URCB, LCB and LOG. All of these classes will be explained in the following. Also the LN class has attributes for instance name and instance path name. If the logical node is LLN0 (Logical node zero, a special LN class defined in [11]), a few additional classes are also contained in it. The services provided by the LN class are `GetLogicalNodeDirectory` and `GetAllDataValues`. The LN class is defined in clause 9 of [9].

In [11] the concept of logical nodes is further elaborated. 91 compatible logical node classes are defined which are specialisations of the LN class explained above. These classes can be grouped together as shown in table B.1.1 of the Appendix. The 91 compatible logical node classes each describe and represent some particular functionality in a substation. Some examples of logical nodes:

- The measurement logical node which is called MMXU. The first letter indicates that it belongs to the group M (Metering and Measurement). MMXU is used for calculation of currents, voltages, powers and impedances in a three-phase system. It is defined in [11](5.10.7).
- The circuit breaker logical node, XCBR, in group X (Switchgear). It is used for modelling switches with short circuit breaking capability. Defined in [11](5.12.1)
- The alarm handling logical node, CALH, in group C (Control). It allows the creation of group warnings and alarms. Defined in [11](5.6.2).

The compatible logical node classes are defined as subclasses of the LOGICAL-NODE class defined in [9]. Each of them are defined with relevant attributes from the many DATA classes (see below). Obviously, it will be too tedious to list all of the logical nodes and class definitions here. For a complete definition of all compatible logical node classes, refer to [11].

Object Reference	Type	Remark
MMXU1	LN	Measurement LN
MMXU1.PhV	DATA	Phase to ground voltages
MMXU1.PhV.phsA	DATA	Value of Phase A
MMXU1.PhV.phsA.cVal	DataAttribute	Complex Value
MMXU1.PhV.phsA.cVal.mag	DataAttribute	Magnitude of complex number
MMXU1.PhV.phsA.cVal.mag.f	DataAttribute	Floating point number

Table 2.1.2: Example of types: The logical node MMXU. The recursive structure of types DATA and data attributes is illustrated.

## DATA

The DATA class, like the LN class, is a key element of the IEC61850 standard. It is defined in clause 10 of [9]. Values of DATA instances represent meaningful information about substation devices, such as currents, voltages, power, phases, temperatures, status, timestamps and so on. The DATA class is defined in a somewhat more complicated manner than the classes previously explained in this analysis. This is due to the fact that a DATA instance may contain attributes which are themselves instances of the DATA class. Hence, it can be said that the DATA class is recursively defined. Furthermore, one of the attributes of the DATA class may contain the class `DataType` which is also recursively defined. In order to get a profound understanding of the DATA class and its attributes and their types, a thorough study of clause 10 of [9] is recommended.

Like the LD and LN classes, the DATA class has attributes for name and path name of a DATA instance. These are called `DataName` and `DataRef`. It also has an attribute called `Presence` indicating whether or not the instance is mandatory or optional.

The DATA class may also contain a list with a number of attributes (zero or more) called `DataAttribute`. These constitute yet another class which contains attributes for name, path name and presence (optional/mandatory) along with zero or more instances of `CompositeComponent` and zero or one instance of `PrimitiveComponent` (but at least one must be present). The `DataAttribute` class is a subclass of `DataType` and the `CompositeComponent` class is also a subclass of the `DataType` class which means that yet another recursive structure is found here.

As an example to illustrate the structure of the DATA class, the measurement logical node class MMXU may be considered. It represents measuring functionality in a substation. A logical node instance can be named `MMXU1`. In table 2.1.2, some possible instances of DATA and `DataType` are listed.

Data attributes of type `DataAttribute` can be classified according to their specific

use. The IEC61850 standard defines a number of *functional constraints* which are attributes of each data attribute. A functional constraint indicates that the data attribute is used for some particular purpose, such as reporting, logging, configuration, setting groups and so on. The functional constraints of a DATA instance decide the rights of services to read and/or write the DATA. The complete definition of functional constraints is found in [9].

## 2.1.7 Information Exchange

### DATA-SET

A DATA-SET is an ordered group of ObjectReferences of DATA or DataAttributes. It is defined in clause 11 of [9]. The data and data attributes are organized into a DATA-SET for the convenience of the client. The DATA-SET class contains attributes for name, path name and a list of dataset members which contains functionally constrained data or data attributes of the data and data attributes which are organized in the DATA-SET. Both the client and the server shall keep track of the order and membership of a DATA-SET, so that only the name of the DATA-SET and the current values need to be transmitted.

There are five services provided by the DATA-SET class. The membership and order can be retrieved with service `GetDataSetDirectory`. DATA-SETs may be created and deleted by `CreateDataSet` and `DeleteDataSet`. Finally, values may be read and written by `GetDataSetValues` and `SetDataSetValues`.

### Substitution

The substitution model (defined in clause 12 of [9]) allows for a value of a data attribute to be substituted with a manually entered value if the attribute has a particular functional constraint, namely MX for analogue values or ST for status values. The process relies on the substitution-related data attributes defined in [10], like `subEna` and `subVal`. This allows a client operator to enter a substitution value for particular data attributes, so that if these are retrieved, for instance by a report, the substitution value will be transmitted instead of the value determined by the process.

### SETTING-GROUP-CONTROL-BLOCK

The SETTING-GROUP-CONTROL-BLOCK class model is defined in clause 13 of [9] and is abbreviated SGCB. It allows for a DATA instance to have several values, which are used one at a time. It can also be applied in this manner on a set of DATA instances. The SGCB can provide one or many setting groups (SGs) which contain

values for each of the relevant DATA instances. At any given time, only one of the SGs is active and therefore the DATA instances have values corresponding to that SG. The DATA must be properly functionally constrained in order to be used by services of this class.

## Reporting

Reporting is handled by the REPORT-CONTROL-BLOCK class of the ACSI, defined in clause 14 of [9]. This class shall control the procedures that are required for reporting values of DATA from one or more LNs to one client. Three trigger options are defined, namely data-change, quality-change and data-update, which can cause a report to be sent to a client. Two classes of report control are defined, BUFFERED-REPORT-CONTROL-BLOCK (abbreviated BRCB) and UNBUFFERED-REPORT-CONTROL-BLOCK (abbreviated URCB).

The class BRCB allows for the sending of reports to be issued immediately, or for the events to be buffered for transmission after an amount of time specified in the `bufTm` property. Furthermore, BRCB provides the sequence-of-events (SoE) functionality and if the connection is broken when reporting is to take place, the report is buffered and sent when the connection is re-established.

The class URCB on the other hand, only allows transmission of reports according to the time specified in the `bufTm` property. No further buffering is performed if the connection is lost and the reports are discarded in that case. URCB does not provide SoE functionality.

For both types of reporting, the server must restrict access to an instance of a report control block to one client at a time. The client will be associated with the control block and that client will be the only one receiving reports from the control until the association is released or aborted. In order for more than one client to receive reports of the same values of DATA, multiple instances of the report control block classes must be made available. It is also defined in the standard how this should be achieved. In this context, it must be discerned between buffered reporting and unbuffered reporting.

In the case of buffered reporting, it is important that a client whose connection is lost in the middle of the transmission of the report, is associated with the same report control instance the next time the client connects. This way the report control can keep track of which report was successfully transmitted last, and thus, which reports are yet to be transmitted. For unbuffered reporting, this is not necessary. The class provides services for sending a buffered report and reading or writing attributes of a

BRCB.

The report control class for unbuffered reporting, URCB, is somewhat similar to the BRCB class.

## Logging

Class models for logging are defined in clause 14 of [9]. The purpose of logging is to maintain an internal storage of historical data values for subsequent reviewing or producing statistics. It is independent of communication and should thus take place even if communication breaks down. Logging can be generally divided into periodic recordings and event-triggered SOE data. There are two classes that handle logging in the IEC61850 standard; the LOG-CONTROL-BLOCK class (abbreviated LCB) which controls the logging and the LOG class. One LOG may be controlled by multiple LCBs.

The LCB class controls the procedures that are required for storing values of DataAttribute into a LOG. Each enabled LCB shall associate a DATA-SET with a LOG. Changes in a value of a member in the particular DATA-SET will be stored as a LOG entry. LCB contains attributes for name, path-name, whether the instance is enabled, DATA-SET being monitored, optional fields, trigger options and integrity period. Furthermore there is a LogRef attribute indicating the reference to the LOG to which log entries shall be recorded. Two services are offered, GetLCBValues and SetLCBValues. As the names indicate, the first one retrieves attribute values from the LCB while the second one sets the attribute values.

The LOG class contains the actual entries and shall be filled on first-in first-out (FIFO) basis. That is, when the stored data reaches the maximal size of the log, the oldest entries shall be overwritten. The LOG class has attributes for name and path-name of LOG instances. It also has attributes with timestamp for the oldest log entry time and newest log entry time.

The entries are stored in the Entry parameter, with attributes for entry time, entry identifier, and parameter EntryData with dataset, value and trigger options. The services defined for the LOG class are:

- QueryLogByTime: read log entries between selected start time and stop time
- QueryLogAfter: read log entries selected by entry ID
- GetLogStatusValue: get the status value of a log

## **Generic Substation Event - GSE**

The purpose of the GSE class model is to provide the possibility for fast and reliable system-wide distribution of input and output data values. It is intended to provide an efficient method for delivering the same generic substation event information to physical devices through multicast/broadcast services. The GSE class model is defined in clause 15 of [9]. The details of how reliability and a short transmission delay are achieved depend upon the SCSM (see below) and the communication stack used.

## **Transmission of sampled values**

Clause 16 of [9] defines a model for transmission of sampled values. This model applies to the exchange of values of a DATA-SET of the common data class SAV (defined in [10]). The exchange is based on a subscriber/publisher mechanism and takes into account time constraints and sampling rates. Two methods are provided, namely MULTICAST-APPLICATION-ASSOCIATION (using multicast sampled value control, MSVCB) and TWO-PARTY-APPLICATION-ASSOCIATION (using unicast sampled value control, USVCB).

## **CONTROL**

The control model, defined in clause 17 of [9], provides services for a client to control DATA related to external devices and control outputs. This is done by operating on DATA with certain functional constraints (CO or SP) and of certain common DATA classes (SPC, DPC, INC, BSC, ISC or APC). DATA of one of these classes are called control objects in this context.

The following services are defined in the control class model:

- **Select (Sel) / SelectWithValue (SelVal)**  
The Select service only selects an object to be controlled. The reference to the object is the only parameter and it is also included in the response. The SelectWithValue service includes a value for the object to be controlled, the time stamp for when the control request is sent, whether the control is part of normal operation or a test, and finally which checks shall be performed by the control object before operations are issued.
- **Cancel**  
The Cancel service is used for deselecting an object. The parameters included are a reference to the object, a time stamp and whether it is a test.
- **Operate (Oper) / TimeActivatedOperate (TimeOper)**  
These services require exactly the same parameters as SelectWithValue. In the

TimeActivatedOperate, the time stamp is the time when the operation shall be issued. The service checks for validity of this time stamp.

- **CommandTermination (CmdTerm)**  
Terminates the command. Parameters are reference to control object, time stamp and whether it is a test.

The behaviours of the services are defined in state machines and there are four behaviour models defined. These are listed below along with remarks to their use.

- **Direct control with normal security (direct-operate)**  
used for operations on local DATA or on DATA that influence external devices where return information is not supervised. Uses services Operate and TimeActivatedOperate.
- **SBO control with normal security (operate-once or operate-many)**  
SBO means "Select Before Operate" and uses services Select, Cancel, Operate and TimeActivatedOperate. In this case, the control object checks if the client has the appropriate access authority and that the object is not currently selected before operation is carried out.
- **Direct control with enhanced security (direct-operate)**  
Uses Operate, TimeActivatedOperate and CommandTermination. In the case of enhanced security, each command sequence shall be terminated with the CommandTermination service, and additional supervision of the status value is ensured by the control object.
- **SBO control with enhanced security (operate-once or operate-many)**  
Uses services SelectWithValue, Cancel, Operate, TimeActivatedOperate and CommandTermination. In this case, the Operate request must be issued before a deselect timer expires, and the state of the control object must be Ready for that particular client for the operation to be carried out.

## **Time and time synchronization**

Clause 18 of [9] defines the time and time synchronization model which shall provide the UTC<sup>5</sup> synchronized time to applications in the server and client substation IEDs. The model comprises external information from an external source, a time server, a time synchronization protocol, time stamp semantics, presentation of time stamps, and the server and client involved. Some of these are defined in IEC61850 while others may depend upon the SCSM.

---

<sup>5</sup>Coordinated Universal Time



## Naming conventions

Clause 19 of [9] defines conventions for class and instance naming. The general format of an instance name follows the pattern:

```
LD/LN.Data[.Data[. ...]]DataAttribute[.DAComponent[. ...]]
```

where the inner bracket indicates further recursive definitions of nested data attribute components. As an example,

```
E1QA5/XCBR.Pos.ctlVal
```

may refer to either a class definition or an instance of the class. On the other hand,

```
E1QA5/XCBR8.Pos.ctlVal
```

can only refer to an instance since the logical node name contains an 8 which the class name cannot do. The names also have to comply with certain length definitions.

## File transfer

The ACSI shall provide a file store according to the file model of clause 20 of [9]. It contains a class definition for FILE with services for

- retrieving the contents of a file from the server to a client
- sending the contents of a file from a client to the server
- deleting a file in the file store of the server
- retrieving the file name and attributes from the file store

### 2.1.8 Communication

The parts up to and including 7-4 of the IEC61850 standard describe how a substation and related components are modelled in a piece of software with the capability of controlling and monitoring the substation. In part 8 and 9, focus is shifted to the larger picture, involving clients possibly residing on other computers than the server, but connected in a network to the server. In order to make this feasible, a communication structure must be decided upon, so that the client and the server can 'understand' each other.

This means that the abstract services defined in the ACSI of [9] must be made available in some protocol that the server and clients are able to apply. The term abstract indicates that only aspects required on the receiving side of a request are included. Thus, the ACSI defines only the semantics, not the syntax, of the services.

In order to make these services available for a client, the syntax is also required. This could be achieved in a number of ways. For instance, a communication protocol could be invented as part of the IEC61850 standard. However, to achieve interoperability and make it easier to employ the new standard, it would be favorable to choose an existing and widespread communication structure and provide a *mapping* to one or more of these protocols. This is probably the reason why the IEC have chosen to provide mappings to already standardised protocols such as MMS and Ethernet. Such mappings are called Specific Communication Service Mappings and are abbreviated SCSM.

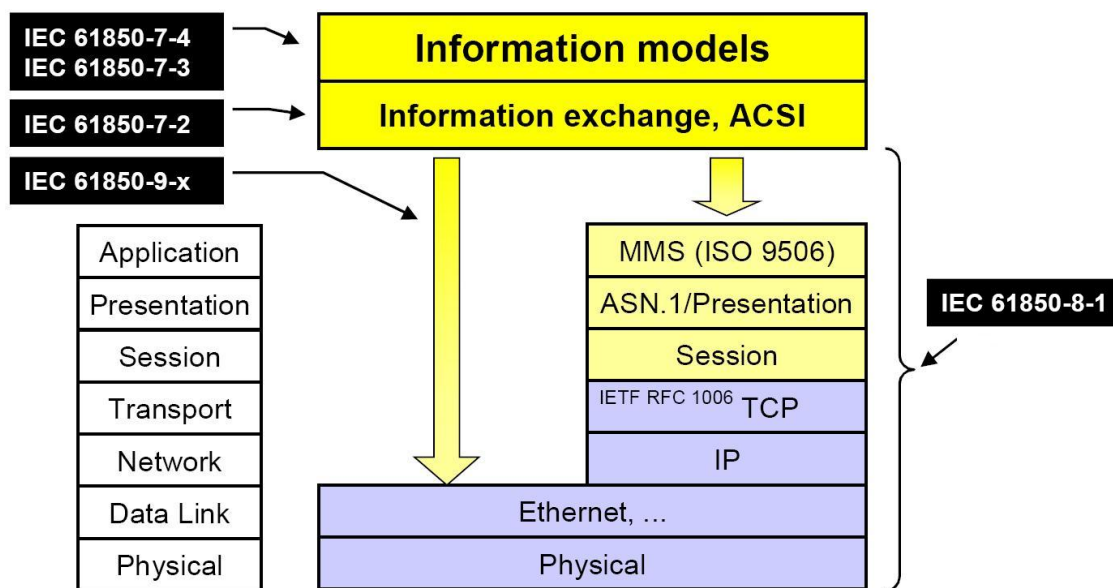


Figure 2.6: The SCSMs of IEC61850 placed according to the OSI layers [8]

Figure 2.6 illustrates the three SCSMs proposed in the IEC61850 standard and places them according to the OSI model. In the figure, the SCSMs are named according to their part numbers in the standard, IEC61850-8-1 ([12]) and IEC61850-9-x for IEC61850-9-1 and IEC61850-9-2 ([13] and [14]).

The SCSM of IEC61850-8-1 maps most ACSI services to the Manufacturing Message Standard which is an international networking standard (ISO 9506), usually abbreviated MMS. It also provides mappings for Sampled Values, GOOSE, Time Synchronization and GSSE to Ethernet. This SCSM thus covers most of the services explained in this chapter. MMS is placed in the application layer of the OSI model while TCP/IP and Ethernet are placed in the so-called T-profile, comprised of the transport layer and those beneath it. Ethernet is also an international standard (IEEE 802.3).

Ethernet is also the basis for the SCSMs proposed in IEC61850-9-1 and IEC61850-9-2. IEC61850-9-1 is named "Sampled values over serial unidirectional multidrop point to point link". As can be seen from the figure, this SCSM is placed only in the T-profile while the A-profile layers are not specified. In this SCSM, a mapping is provided for the communication between merging units and devices on bay level which need raw data [13].

IEC61850-9-2 is named "Sampled values over ISO/IEC 8802-3" and is intended as a supplement to IEC61850-9-1 to provide a complete mapping for sampled measured values [14].

## 2.2 Analysis of Scenarios

This section gives an analysis of the scenarios provided by Brodersen Controls A/S. Each subsection begins with explaining the scenario itself. Afterwards there is an analysis of the IEC61850 standard with respect to the current scenario. The analysis shall identify components of the ACSI which must be implemented in order to provide functionality for the relevant scenario. The analysis shall place the scenarios in relation to the scope of the IEC61850 standard.

Annex G of IEC61850-5 [6] is used for some cases, since it contains a number of scenarios for which logical nodes are listed.

### 2.2.1 Scenario 1: Event based single alarm

*A monitor relay on the secondary side of a transformer detects a low voltage level. The level detection sets a physical input on the RTU. The RTU application program detects that the input is activated and adds immediately a time stamp with millisecond accuracy to the input event. The alarm is now reported to the SCADA system with highest priority. Any other derived alarms from the actual phase break event (maybe 100 alarms) should be suppressed until the highest priority alarm has been sent. The alarm should be sent including identification, time stamp and cause of transmission, The RTU should keep the alarm data until it is acknowledged by the SCADA system.*

In order for this case to be covered by the IEC61850 server, it would be ideal to map the monitor relay to the logical node PTUV which represents a protection relay which operates when its input voltage is less than a predetermined value. PTUV is defined in clause 5.4.24 in [11]. The minimal allowed value of the voltage can be stored in `PTUV.StrVal.minVal`. How the monitor relay is mapped to the logical node in the

data model on the RTU is outside the scope of the standard and must be handled by an extra configuration file or module.

It is necessary to distinguish between alarms and reports in this context. In IEC61850 terms, an alarm is just an event with time stamp, and this event can be reported by the server to the client. In order for such an event to be sent as an alarm in the sense of the scenario, the `bufTm` property for the report control block should be set to zero, so that the report would be transmitted immediately. However, the standard does not include a way of giving priority to alarms in such a way that a large number of other reports can be suppressed until one particular report is sent. This would also have to be handled by the extra RTU functionality.

A report can be generated based on trigger events in a dataset. In this case, a dataset would be created containing some of the data objects or data attributes in the PTUV, such as `PTUV.Op.General` which indicates whether the protection relay operates. The report is then set up to subscribe to this dataset. When the voltage is detected to be too low, the protection relay will operate and elements in the subscribed dataset will change, thereby triggering the event which causes the report to be sent.

The reporting services of classes URCB and BRCB are obviously required in order to implement this scenario. Each of these control blocks has 3 services, a report service, a get service (`GetBRCBValues` or `GetURCBValues`) and a set service (`GetBRCBValues` or `SetBRCBValues`). The report service sends the report and is the most essential requirement for this scenario.

The get service allows the client to view what the report subscribes to and the set service can edit the subscriptions of the report. Therefore, these services are also highly relevant for the scenario to allow the client to review or change report settings.

If the client is to be presented with an overview of the structure of the substation prior to viewing or editing report settings, some more basic services are also required such as `GetServerDirectory`, `GetLogicalNodeDirectory` and so on.

Association services are also a basic requirement in order for a client to connect to the IEC61850 server. There are three association services, `Associate`, `Abort` and `Release`.

The basic services for retrieving an overview of the substation components (`GetServerDirectory` etc) and association services can be seen as a requirement in all the scenarios and will not be mentioned for each case.

The IEC61850 standard does not prescribe how the particular inputs shall be mapped to elements in the data model, such as the data attributes in the logical node PTUV.

## Sequence Diagrams

Figures 2.7, 2.8 and 2.9 show sequence diagrams for some of the services necessary for scenario 1. The role of a system sequence diagram is to visualise the interaction between components of the system, in these cases the client and the server. In this chapter, system sequence diagrams show the client and the system and messages that are sent (invoked services) as well as responses. The diagrams shown here include parameters that are sent in both directions. When the parameter names are long, they are shown abbreviated in the diagrams. The ACSI services in IEC61850 always define one response for normal operation and also an error response for failure. The diagrams will show only the normal operation responses. In chapter 3 the sequence diagrams will be elaborated and used for designing solutions for the cases.

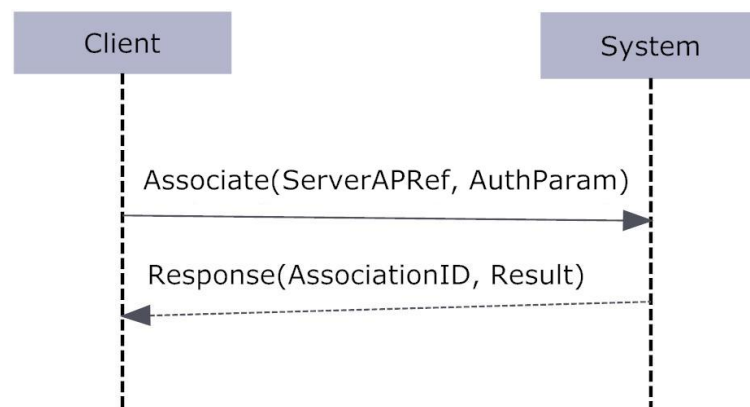


Figure 2.7: Sequence Diagram for Association. The parameters sent from the client are ServerAccessPointReference and AuthenticationParameters. In case of error, an error response is sent

Association can be thought of as the initiation of communication so this will take place before any of the other services become relevant. Therefore, this service is depicted in a sequence diagram in figure 2.7. A service such as GetServerDirectory might be requested in order for the client to become acquainted with the structure of logical devices on the server. Subsequently, the client could send requests for services like GetLogicalDeviceDirectory, GetLogicalNodeDirectory and so on, to retrieve a full overview of the substation structure residing on the server. Sequence diagrams for these services would resemble the one in figure 2.8. The SetURCBValues service, visualised in figure 2.9, allows the client to set up unbuffered reporting of changes in values of data and data attributes.

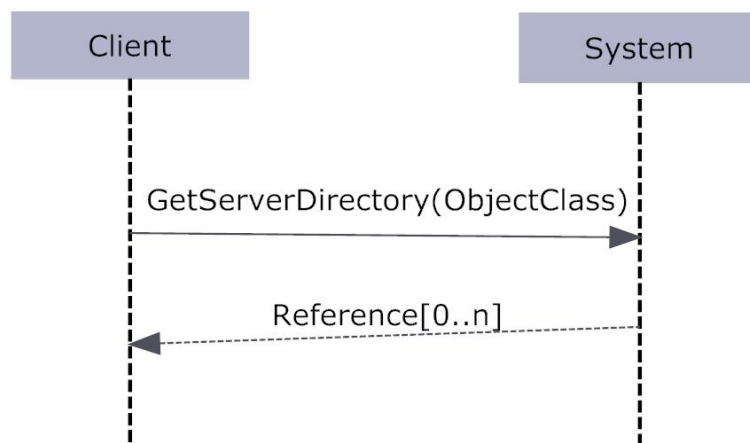


Figure 2.8: Sequence Diagram for GetServerDirectory. The client can choose either File or Logical Device as ObjectClass.

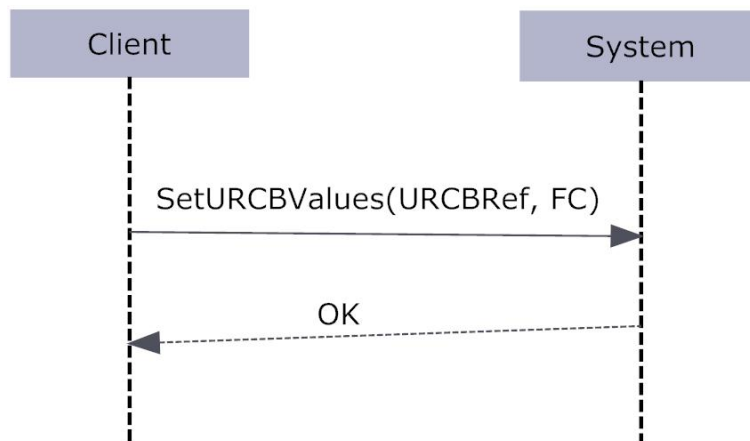


Figure 2.9: Sequence Diagram for SetURCBValues. The client sends parameters URCBReference, FunctionalConstraints and up to 10 other optional parameters

### 2.2.2 Scenario 2: Event based double alarm

*A safety control relay on a transformer is monitored by a double input. The safety relay has two outputs for monitoring of state. Input 1 is activated and input 2 is not activated when relay is ON. When relay is switched to OFF, input 1 is switched to OFF and input 2 is switched to ON. If relay is not switched properly input 2 is not activated and none of the inputs are activated. This state is not allowed and has to be reported as an immediate state which is considered "Not allowed" and is reported at the SCADA as relay failure.*

Since this scenario involves many of the same aspects as scenario 1, most of the same functionality is needed. It would be ideal to map the case to a LN for switching since

that is essentially what is being performed in this case.

An important aspect in this case is that the state of the relay is monitored by two inputs instead of one, and that it is necessary for the data attribute for the state to be able to represent a bad state as well as on/off.

As mentioned in the previous scenario, the way inputs are mapped to elements in the data model are outside the scope of IEC61850. The important thing is that they are mapped to an element which can represent a bad state. This is achieved by data class DPC (Controllable double point, defined in 7.5.3 of [10]). This data class contains a status value which can assume the four values `on`, `off`, `intermediate-state` and `bad-state`.

The data class DPS for Double Point Status also contains such a value, but DPS is not included in any of the predefined logical nodes of [11]. DPC on the other hand is part of the circuit switch LN, XSWI. It would therefore be ideal to have XSWI implemented and have the two monitoring inputs mapped to the data attribute `XSWI.Pos.stVal`.

The same services are required for this case as for the previous one.

### **2.2.3 Scenario 3: Control physical output pulse on RTU**

*From the SCADA it should be possible to control a physical output relay on the RTU. The output is activated as a pulse with a duration of 5 seconds. The control message from the SCADA should be assigned a system time stamp, and the RTU should verify that the control message has a valid time - e.g. not older than 120 seconds. If the message is older than 120 seconds the control message has to be ignored. The complete control procedure should include a "select and execute" procedure in order to ensure that you only control output from the SCADA that was really intended to be controlled.*

The Control model defined in clause 17 of [9] adds five services for controlling outputs. These are explained in 2.1.7 and are the ones needed for controlling an output with select-and-execute procedure or select-before-operate (SBO) which is the term used in the standard. The control model also allows for a deselect timer to be set to e.g. 120 seconds.

One of these services, `TimeActivatedOperate`, provides the possibility to set a timestamp for when the control command shall be issued. However, none of the services include parameters for the duration of the control, in this case the duration of the pulse. Therefore, the ACSI does not include functionality for the duration of a com-

mand.

Such an issue could be handled by the piece of software mapping of IEC61850 objects to inputs and outputs on the RTU. For example, some operations could be defined to always last e.g. 5 or 10 seconds. But the duration could not be sent as a parameter in requests to services of the standard.

It is also possible to imagine that the functionality can be implemented in the client. This functionality could be composed of invocations of `SelectWithValue`, `Cancel` and `CommandTermination`, but the client would have to wait for the proper amount of time before issuing the control commands deactivating the output again. Such a solution could lead to inaccurate durations due to delays in transmission.

This scenario does not relate to a specific type of logical nodes.

#### **2.2.4 Scenario 4: Send alarm setpoint to RTU**

*From the SCADA it should be able to set an alarm setpoint to level alarm control of analogue inputs. The alarm setpoint is in this case an engineering 14 bit value used internally in the RTU for setting properties for level alarms for e.g. voltage, current and power measurements realized via physical analogue inputs on the RTU. In general the same requirements for time stamp evaluation and "select and execute" has to be considered.*

In order for this scenario to be implemented, the analogue inputs shall be mapped to LNs containing data objects of type MV (measured value) or CMV (complex measured value). These data objects contain the attribute `db` (deadband) which can be set to a proper value. The deadband makes sure that the magnitude of the input signal, the `mag` attribute which is an analogue value, is not changed unless the instantaneous magnitude `instMag` changes by a margin greater than the value of the deadband.

The MV data class also has an attribute called `rangeC` of type `RangeConfig`, which allows a `min` and `max` to be set. These can then prescribe the range for which the signal is inside bounds and thereby what values shall cause an alarm to be sent.

#### **2.2.5 Scenario 5: Event report of analogue input**

*Based on some predefined threshold values in the RTU, analogue floating point measurements from analogue input on the RTU should be reported to the SCADA. The threshold value should e.g. be a percentage value of the analogue measurement. When*



*the analogue measurement changes more than the defined threshold since last reported value, the input value should be sent to the SCADA with time stamp.*

This scenario requires exactly the same functionality as the previous one. The deadband value is set in units of 0.001% and can be set with values from 0 to 100 000. That is, if a threshold of 5% of the measured value is desired, the deadband value can be set to 5000.

## **2.2.6 Conclusion to Analysis of scenarios**

This analysis has outlined some services, logical nodes and data which should be included in the implementation of the IEC61850 server in order for the requirement of the scenario to be fulfilled. A fair amount of extra functionality has been identified which is not immediately part of the IEC61850 standard, but which would be needed in order for the IEC61850 server to be taken into practical use and handle the scenarios mentioned. This includes functionality providing an operator with the possibility to map inputs or combinations of inputs to elements in the IEC61850 data model, specify the duration of a control of an output and assign priorities to event reports. To conclude, the analysis of the scenarios has revealed that the standard and the scenarios do not deal with exactly the same issues.

## **2.3 Analysis of the Brodersen RTU32**

In order to put the software to use on the Brodersen RTU32, special attention needs to be given to the environment on which the software shall ultimately run. Therefore a brief analysis of the RTU will be given in this section.

The idea of using an RTU in a project like this is based on the fact that an RTU may represent the substation in a communication network. The IEC61850 server running on the RTU shall therefore contain a model of the substation with objects representing the components in the substation. The RTU shall be connected to the substation components through the RTU's inputs and outputs, and the software on the RTU shall keep track of the structure of the substation and its components, measurements of values of data related to these components and also the software shall carry out proper operations depending on changes in these data.

The RTU32 has a database in a DLL file which is connected with the I/O ports. This file is called `wt00132.dll`. It is possible to read and write from and to the I/O ports by use of e.g. Visual Basic applications issuing requests to the DLL. This DLL shall be the means of all real-time communication between the substation and the IEC61850

Server. In addition to this communication, preconfiguration shall be possible by the use of an SCL file.

The RTU32 runs Windows CE 5.0 which is a minimalistic version of Windows. This does not mean that it is the "normal" Windows, with some functionality taken away. Windows CE is an altogether different operating system with a different kernel compared to other Windows versions, however it has some similarities to more common versions of Windows. Windows CE is optimized for computers with minimal storage such as an RTU.

On Windows CE, the .NET Compact Framework (.NET CF) must be used. Version 2.0 of 2005 is part of the RTU setup. The Compact Framework is a down-scaled version of the full .NET Framework which implements approximately 30% of the full .NET Framework Class Library [19]. As a consequence, a considerable amount of functionality is not available on the RTU such as hosting web services, remoting, SOAP serialisation, binary serialisation and some socket options. These shortcomings must be kept in mind in the design process and will make the implementation more cumbersome than if it were performed on a full .NET Framework.

## 2.4 Specification of Requirements

Based on the previous sections, where the IEC61850 standard, the scenarios and the RTU have been analysed, a specification of requirements shall be prepared for the design and implementation of the basic IEC61850 server. Because of the vast functionality and considerations included in the standard, the goal of this project cannot be to produce a complete IEC61850 compliant implementation.

As the analysis of scenarios provided by Brodersen Controls A/S showed, there are a number of requirements suggested in the practical application of the RTU which are not dealt with in the standard. The requirements for the implementation of the basic IEC61850 server of this project must therefore be made carefully and with consideration given to the standard.

The fact that the solution shall run on a compact framework under Windows CE also makes it necessary to consider carefully which requirements can be specified, since a larger portion of the implementation must be made "from scratch" than would be the case on a normal .NET framework.

It has been chosen to prepare the specification of requirements based on the desire to implement a *basic IEC61850 server*. Priority shall therefore not be given to some of

the RTU specific functionality suggested in the scenarios but on basic functionality of an IEC61850 server. This requires a basic implementation of the data model and basic services. Furthermore, it shall be attempted to make the implementation generic so that the server can later be extended to cover the whole IEC61850 standard and also RTU-specific functionality as mentioned in the scenarios.

The following requirements have been chosen:

#### **Specification of Requirements**

- The solution shall implement a basic IEC61850 server which is able to run on a Brodersen RTU32 under Windows CE
- It shall be possible to read a configuration file in SCL format for a substation and have a data model with a server instance generated on the basis of the SCL file
- A client shall be able to connect to the server and request basic ACSI services such as reporting and logging
- The solution should be generic so that extension is possible later

Implied in these requirements is a delimitation of the focus of the project. It is not required, for instance, that the communication between the client and the server complies with the SCSMs proposed in the IEC61850 standard.

It is not required that all conceivable cases shall be handled, that is, not all logical nodes, data and services shall be implemented. This is a necessary delimitation because of the comprehensive nature of the standard. Furthermore, some of the functionality suggested in the scenarios provided by Brodersen Controls A/S is not included in the requirements since they are outside the scope of IEC61850 and would make the focus of the project too unclear.

## **2.5 Conclusion**

In this chapter, an analysis has been presented of the IEC61850 standard, the scenarios provided by Brodersen Controls A/S and the Brodersen RTU. The analysis of the standard has given an overview of the basic concepts of the IEC61850 standard and a basic understanding of the classes involved in the comprehensive model which the standard is based on. The classes of the Abstract Communication Service Interface with information models and information exchange models have been explained briefly. The specific communication service mappings, SCSMs, of the standard have also been explained as well as the substation configuration description language, SCL.

The analysis of the scenarios has placed the RTU functionality desired by Brodersen Controls A/S in relation to the standard. Logical nodes and services required for the scenarios have been identified as well as other functionality that needs to be implemented in order to meet the requirements of the scenarios. Part of this functionality is outside the scope of the standard.

The analysis of the RTU32 has outlined the possibilities and limitations of the RTU on which the implementation shall run. Based on these analyses, a specification of requirements has been given. The requirements focus on implementing a basic IEC61850 server with a data model and basic ACSI services available. Out of scope requirements suggested in the scenarios are not included. Also, the SCSMs of the standard are not included.

To conclude this chapter, this analysis has led to the point where design and implementation can commence. This will be the subject of the next two chapters.

# DESIGN

---

In this chapter, the design of the solution will be presented. First, the overall architecture of the chosen solution will be presented and then the design of the different parts and modules will be explained. The design will be made with the analysis as a starting point. In particular the specification of requirements is important to be kept in mind during the design proces. The solution must be designed to meet the requirements of 2.4. Thus, a design must be proposed for the following:

- A basic IEC61850 server which is able to run on a Brodersen RTU32 under Windows CE
- A mechanism for initialising and configuring the system by use of an SCL file
- A client/server connection with which allows a client to request IEC61850 services such as reporting and logging

Many requirements for services have been visualised by the use of sequence diagrams in chapter 2 and some of these diagrams will be elaborated throughout this chapter to illustrate the ideas behind the design proces.

## 3.1 Architecture of Solution

In this section an overview is given of the architecture of the chosen solution. This is done by a diagram showing the modules of which the solution is composed. The diagram is shown in figure 3.1. The division into these modules is based on the analysis of chapter 2. Later the modules are explained individually.

The module called Information Model contains the data model of the IEC61850 server and is explained in section 3.2. This module is based on the analysis of sections 2.1.3, 2.1.5 and 2.1.6. The Information Exchange Model is the module where the services of the IEC61850 are contained. The design of the module is based on the analysis of sections 2.1.5 and 2.1.7. This module is explained in section 3.3. Communication to client is handled by the Communication module which is explained in sections 3.4. The device module is explained in section 3.5 and contains classes and methods for

providing basic input/output to and from the RTU. This module is based on the analysis of section 2.3. Finally, the substation module functions as the main component which initialises the system. It communicates with - and uses - the other modules. It is explained in section 3.6.

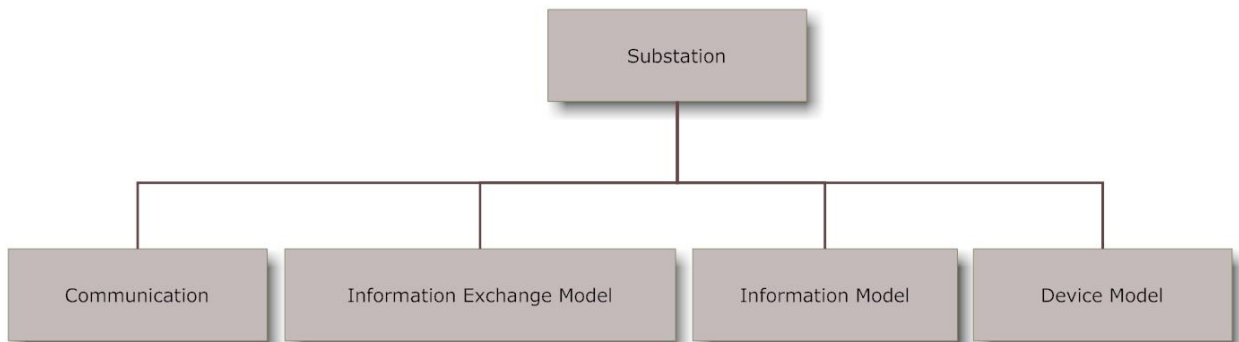


Figure 3.1: Architecture of solution

In the following, each of these modules shall be described through their own UML class diagrams and some of the classes, attributes and methods of the modules shall be briefly explained.

## 3.2 Information Model

The information model of the IEC61850 standard is designed in the module shown in figure 3.2. It is hierarchically structured according to the IEC61850 data model described in 2.1.3. The topmost object in the data model is the Server which is contained in an IED in a substation (see section 3.6). There may be zero or more servers in an IED, typically at least one. Each server contains one or more access points, and an IEC61850 data model comprised of logical devices, logical nodes, data and data attributes.

The Server class may be seen as a collection of the objects below it in the data model. In compliance with the data model, it also has fields for files, associations of clients and access points. The Server also has methods for creating all the objects it contains, i.e. logical devices, logical nodes and so on. Thus, all of these objects are initially created by the Server class. The Server class contains a sorted list of object references and corresponding references to instances of the objects. This allows all objects of the data model to be accessed from the server, given the object reference of the object. This is achieved by the method `getReferencedObject`. Methods are also provided for removing references.

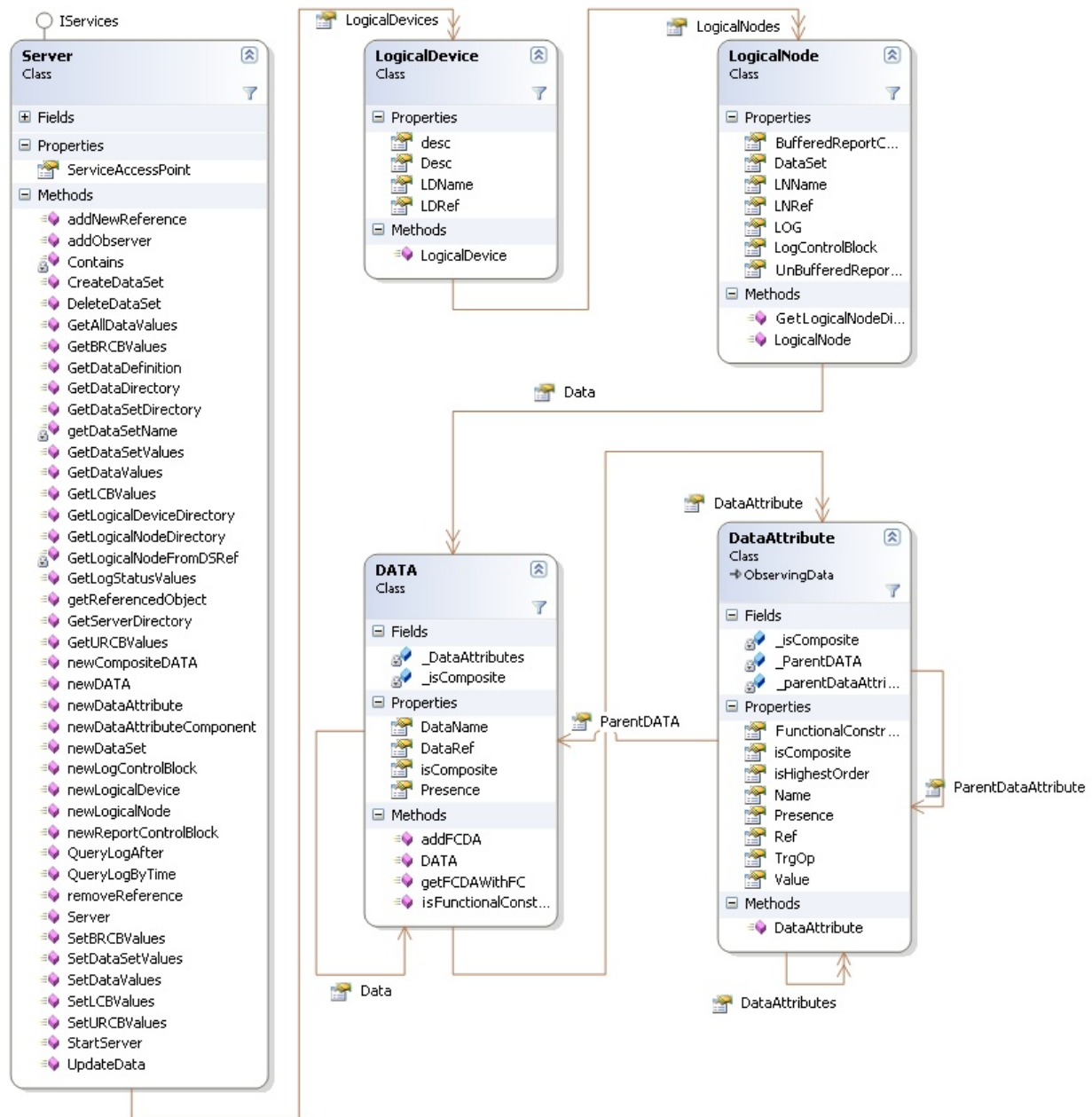


Figure 3.2: UML Class Diagram for the information model module with attributes, properties and methods

When making the design of the data model it is necessary to decide whether the various logical nodes should be designed as individual classes or just instances of the same logical node class. The standard does not actually prescribe rigidly how this should be done.

The refined compatible logical node classes defined in [11] are called specialisations suggesting that they are to be defined as individual subclasses of the abstract logical

node class. However, the definition of the abstract logical node class in [9] contains a list of arbitrary length containing data objects. This means that the refined logical nodes do not as a matter of fact introduce anything new to the logical node object of [9]. And this, in turn, means that the refined classes are not stringent specialisations of the abstract logical node class.

As the class diagram shows, the decision has been made to represent all logical node classes by the same class in the solution, namely the class `LogicalNode`. This has been decided since it allows a generic approach and since it does not limit this implementation notably.

It might be argued that restrictions such as rules regarding mandatory and optional attributes will be far more difficult to handle with the chosen generic approach than would be the case had each logical node been defined in its own class. While this is valid, it is still possible to construct a mechanism which checks for compliance with the rules. Furthermore, the check can also be made before entries are made into the data model as is the case with this solution. This will be explained in 3.7. Finally, it is easily possible to extend the data model with specialisations if it should be desired, so designing the solution with only one logical node class does not limit the possible use of the solution. These considerations are the basis for the design choice of using only one logical node class.

For the data class, the same holds. The abstract data class is defined in [9] and refined in [10]. The abstract data class contains sufficient attributes for it to represent all kinds of data as defined in [9]. Therefore the abstract data class is the only one used in this solution.

Apart from these decisions, the design of the information model is fairly straightforward. The `Server` class contains a list of logical devices as a property, the `LogicalDevice` class contains a list of logical nodes as a property and so on. This is completely in keeping with the hierarchy of the data model as described in section 2.1.3.

As the class diagram in figure 3.2 shows, the `DATA` class has properties for both data and data attributes (named `Data` and `DataAttribute`). This stems from the fact that the `DATA` class is recursively defined, so that an instance of the data class may contain yet another instance of its own class in addition to containing data attributes. A data attribute may also contain a data attribute as a property since it too is recursively defined.

As mentioned in section 2.1.5, some of the services made available by the ACSI are grouped as basic service models. Requests for such services can be handled solely



by use of the data model. These are, for instance, `GetServerDirectory` and `GetLogicalDeviceDirectory`. In figure 3.3, a sequence diagram is shown which visualises these two services. The `TCPSocket` class in the figure is part of the Communication module which is explained in section 3.4.

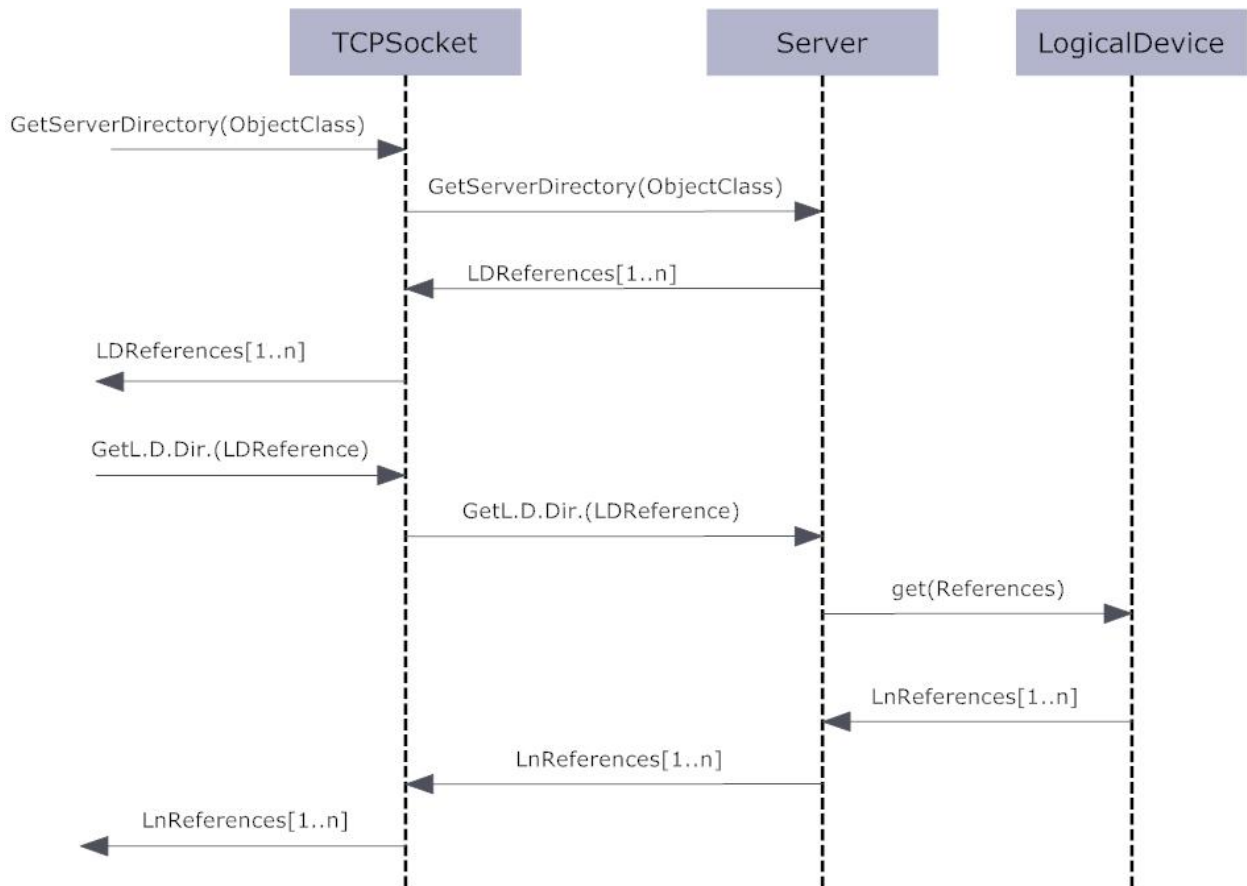


Figure 3.3: Elaborated sequence diagram for services `GetServerDirectory` and `GetLogicalDeviceDirectory`

The logical node class contains not only data, but also datasets and control blocks for reporting and logging which allow events to be reported and logged, according to trigger options. The `Report` and `Log` services require such control blocks and therefore these control blocks are designed in the module called information exchange model (section 3.3).

### 3.3 Information Exchange Model

The information exchange model is the part of the standard that makes services available to a client (other than the basic services) and thus enables a client to monitor changes of values in the data model, review log entries and control outputs among



### 3.3.1 Unbuffered Reporting

BRCB and URCB are control blocks for reporting. The DATASET and control block objects are instantiated in a logical node. If a logical node is declared with an unbuffered report control block, the URCB object is included with a reference to the DATASET object as well as trigger options, a report ID, a report name and other attributes. This shall be governed by the SCL file.

The trigger options can be `dchg` for data change, `dupd` for data update and `qchg` for quality change. Each of these may be set in the SCL file. If `dchg` is set, each change to one of the data attributes of the data object in the contained DATASET will trigger a report to be sent to the client.

This is achieved by an observer design pattern, which is explained shortly. A Report object is created which maintains a list of report ID, time entry and entry data. The entry data is contained in another class, ReportEntryData, which defines a list of data, data attributes and a reason code for inclusion. The URCB collects DATA, triggered by events (changes and updates) for an amount of time defined in the attribute `bufTm`, and for each event, an entry is made into the Report object. After the specified amount of time, the Report is sent to the ReportHandler object which is responsible of transmitting the report to the client.

### 3.3.2 Buffered Reporting

If the logical node instead contains a buffered report control block, the BRCB (instead of URCB) object is included containing a reference to the DATASET object and trigger options. The procedure is similar to that for BRCB until the point after it is sent to the ReportHandler, as will be illustrated shortly.

The procedure for buffered reporting is visualised in a sequence diagram in figure 3.5. This diagram shows which classes are involved in creating a report and how it happens. This part of the procedure would be identical to that unbuffered reporting except that the name would then be URCB.

The ReportHandler shall take into account whether the report is buffered or unbuffered. The further procedure shall make use of the communication module which contains a class TCPSocket which is used for communication with a client. Figure 3.6 visualises this procedure. If there is no connection to the client at the time when ReportHandler attempts to transmit the report, then the report is added to the buffer and ReportHandler attempts to transmit the report again the next time a report is sent to it.

For unbuffered reporting, the report would just be discarded if no connection were available at that time.

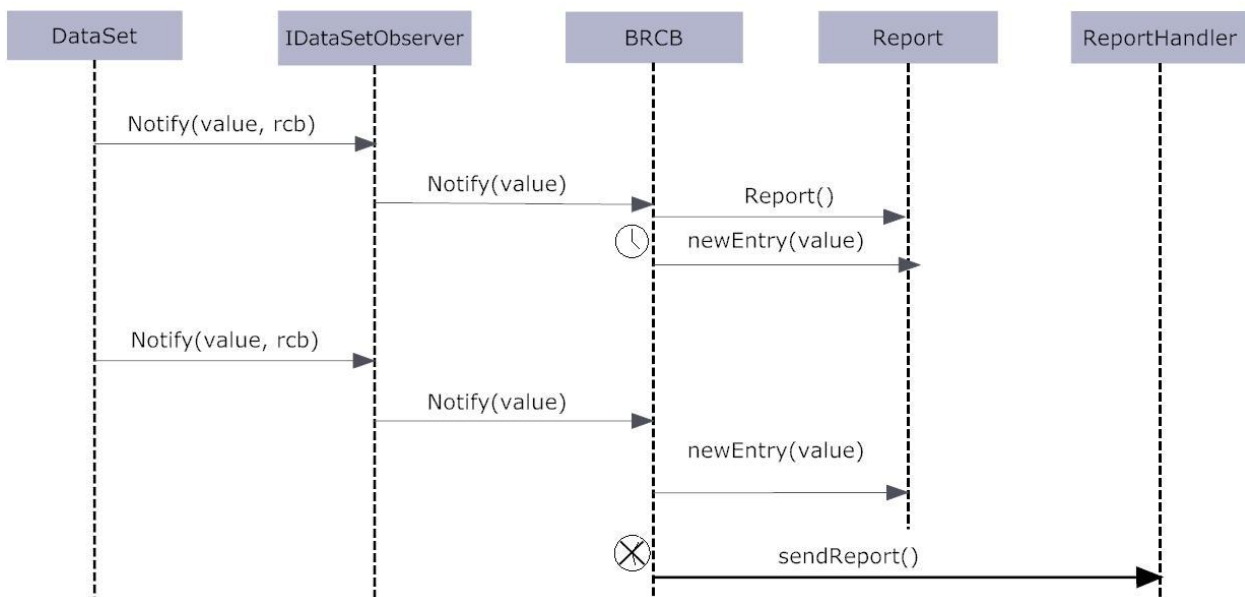


Figure 3.5: Elaborated sequence diagram for reporting prior to server/client communication. The first clock indicates that a timer is started when the new entry is sent to Report. The second clock indicates a time out after which the report is sent to the client. Hold for buffered and unbuffered reporting.

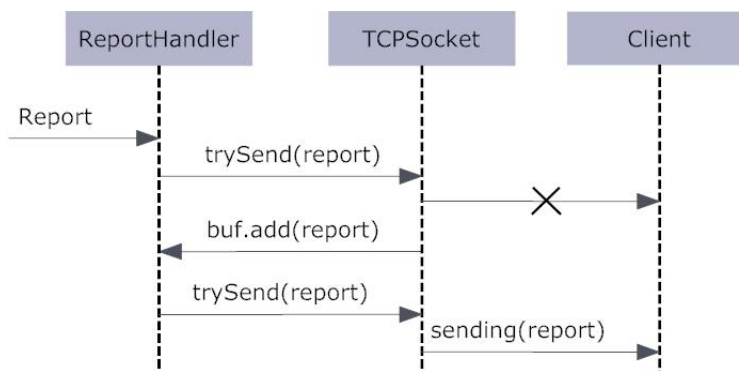


Figure 3.6: Elaborated sequence diagram for buffered reporting from server to client. If no connection is present, indicated by the X, the report is added to the buffer. Holds only for buffered reporting.

### 3.3.3 Logging

LCB is the control block for logging. It is instantiated in the logical node along with the DATASET. If a logical node contains data for logging, it includes a LCB object with a reference to the DATASET. Based on the DATASET object, the observer design pattern notifies the LCB object whenever the relevant data change, and the LCB object causes the new values to be written in the LOG for later review. The data

written to the LOG is defined by the LogEntryData class which contains attributes for the data, data attributes, values and timestamp for each inclusion. The procedure is visualised in the sequence diagram in figure 3.7.

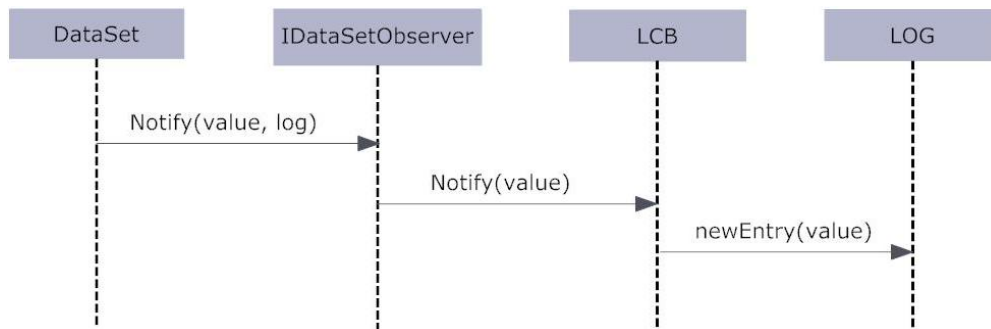


Figure 3.7: Elaborated sequence diagram for logging

### 3.3.4 Observer Pattern

The values in the data model may change at any time as a result of a control services issued by the client or changes in inputs. In order for the data model to be continuously updated according to these changes, an observer pattern is included in the design. This promotes low coupling between the classes of different modules which is useful for a generic approach. Also it avoids the unnecessary amount of extra CPU power it would take to use polling instead of an observer. With the observer pattern, a message is sent when values are changed, so the subscribing object is notified whenever something important happens.

The general observer design pattern is useful in more than one case. Measured inputs may change, and control requests may change data values directly, causing outputs to change afterwards. Furthermore, the report and logging services are to monitor the data model through a DATASET object and shall issue transmissions upon certain changes. To provide the overview of how these scenarios may be solved by using an observer design pattern, the UML class diagram of a *general* observer is shown in figure 3.8.

As the figure illustrates, a concrete observer is able to monitor a concrete subject through the respective abstract classes. This is the general design of the pattern, and in the design of the IEC61850 server, this is used in three cases: the DATASET class monitors DATA classes, the ReportHandler class monitors the RCB class and the LOG class monitors the LCB class.

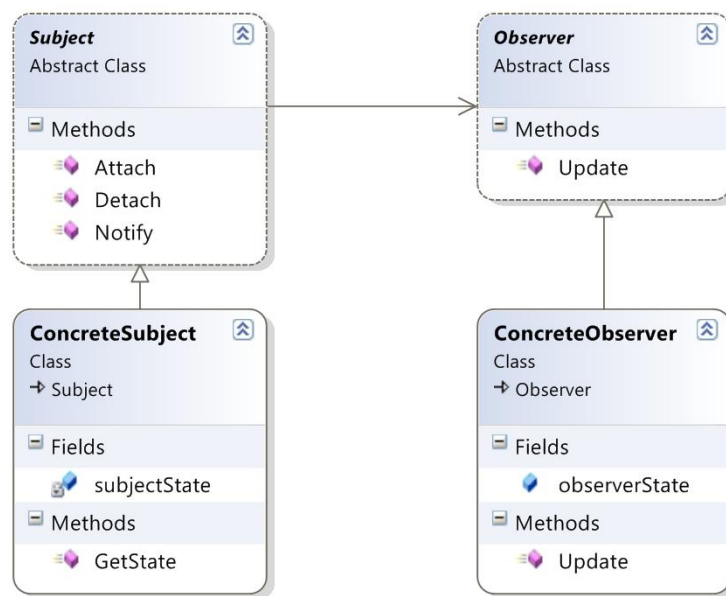


Figure 3.8: UML Class Diagram for the general Observer design pattern

## 3.4 Communication

According to the standard, the server shall be accompanied by a specific communication service mapping (SCSM) supplying the communication platform between the server and clients. The standard proposes mappings to the MMS protocol and to serial unidirectional multidrop point to point link with Ethernet. It has been suggested that web services will be the basis of an alternative SCSM of the standard in the near future.

For this project, priority is not given to the implementation of an SCSM. Therefore it has been chosen to disregard the SCSMs of the standard since they would be a large task to implement. They are actually comprised of other standards, themselves. Web services would have been ideal to use for a project like this, and they are also likely to be incorporated as an SCSM in a future version of the IEC61850 standard<sup>1</sup>. However, the Compact Framework is not intended to be used as a server and therefore does not currently include functionality for hosting web services. Therefore a simpler form of communication is needed, which can make a basic client/server connection possible.

This connection will be provided in the communication module which uses a TCP Socket and sends data back and forth by use of serialisation. The module is shown in the class diagram of figure . As the figure shows, the module is composed of the TCPSocket class, the Envelope class and the ConnectionHelper class. The TCPSocket

<sup>1</sup>This claim is based on the fact that mapping to web services is under progress for the IEC61400-25 standard which is based on IEC61850

object shall listen to incoming requests and direct them to the correct service calls in the exchange model or information model. The interfaces `IReportExchangeModel` and `IExchangeModel` handle this directing. Similarly, the `TCPSocket` class handles transmissions initiated by the server such as reports which shall be sent to the client. In



Figure 3.9: Methods of the communication module

order for the client and server to communicate in the same format, all transmissions are made in a format defined by the `Envelope` class. It has a property for function and a list of parameters. Depending on the function call contained in the envelope, the parameters are parsed accordingly.

Furthermore, with the socket approach, the client and server can only transmit binary data back and forth and they must know beforehand the size of objects which are to be transmitted. The client and server shall both have one `TCPSocket` object running at all times, each with two sockets created. The `TCPSocket` object on both sides shall continuously listen for requests of a fixed size. When the request arrives, it contains the length of the transmission to follow. Having received the length, the `TCPSocket` then listens for an object of the specified length. When the transmission is received, the `TCPSocket` resumes listening for length information.

Furthermore, all data to be transmitted must be converted to binary form. This is accomplished by the `ConnectionHelper` class which uses a compact formatter provided under the LGPL license by Angelo Scotto [21]. The compact formatter is a serialiser which is able to run on the Compact Framework.

### 3.5 Device Model

The RTU module is intended to provide a link between the information model and the input/output of the RTU. It allows an input or output port to be linked to a data attribute in the data model loaded in the main program. This module must further make sure that if an input is connected to a data attribute in the data model, its value must be measured repeatedly with short intervals and changes to the value of the input must be immediately reflected in the corresponding value in the data model. Also it must make sure that if an output port is connected to an attribute in the data model, the output shall be changed immediately upon changes to the value of the attribute. This is achieved through two interfaces, one for each direction. The

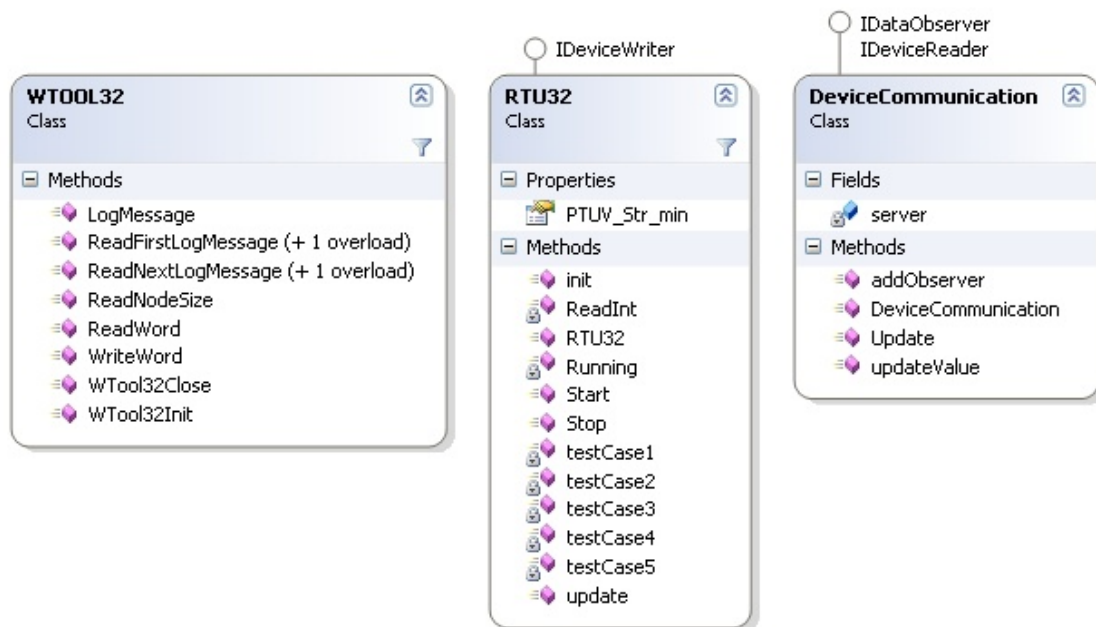


Figure 3.10: UML Class Diagram for the Device module with attributes, properties and methods

RTU32 class contains methods for reading and writing the wtool32 database which is the input and output ports. The reading is performed by RTU32 class which continuously scans the inputs for changes and communicates them to the data model via **IDeviceReader**. When inputs are changed, the RTU notifies the **DeviceCommunication** class which then causes the data model to be updated through **IDeviceReader** interface. The update procedure of the data model is visualised in the sequence diagram in figure 3.11.

When output-related data attributes are changed in the data model, the **DeviceCommunication** class is notified by the observer design pattern. The observer is implemented in class **ObservingData** through interface **IDataObserver**. The **DeviceCommunication** class causes the **RTU32** class to write the correct value to the I/O database.



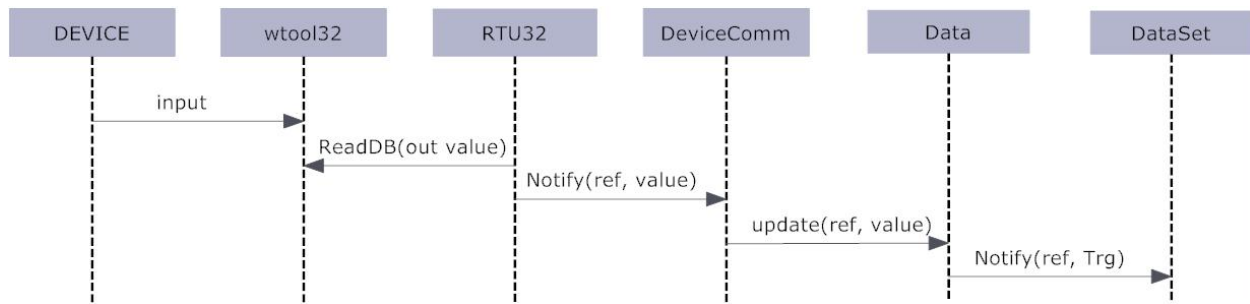


Figure 3.11: Sequence Diagram for update of data model based on changes in inputs. Interfaces to observer pattern are left out, but are represented by `notify()`-calls

This is performed via the IDeviceWriter interface. As soon as the value is written to the database, the output port is physically set correspondingly by the RTU.

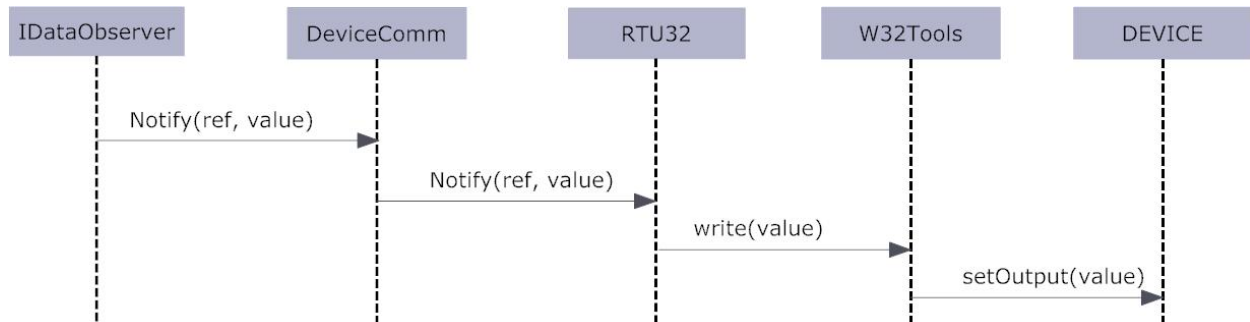


Figure 3.12: Sequence Diagram for update of RTU based on changes in outputs. Interfaces to observer pattern are left out, but are represented by `notify()`-calls

## 3.6 Substation Module

The Substation module initialises the system and is composed of:

- A Substation class initialising the other modules
- An SCLParser class which parses an SCL configuration file for the substation configuration
- An IED class which contains the server

It is illustrated in figure 3.13. The initialisation is based on the parsing of the SCL file. The parser is explained in section [refsec:scldesign](#).

The initialisation causes the parser to read the SCL file given, and generate a full data model based on the contents of the SCL file. It also creates the sockets of the

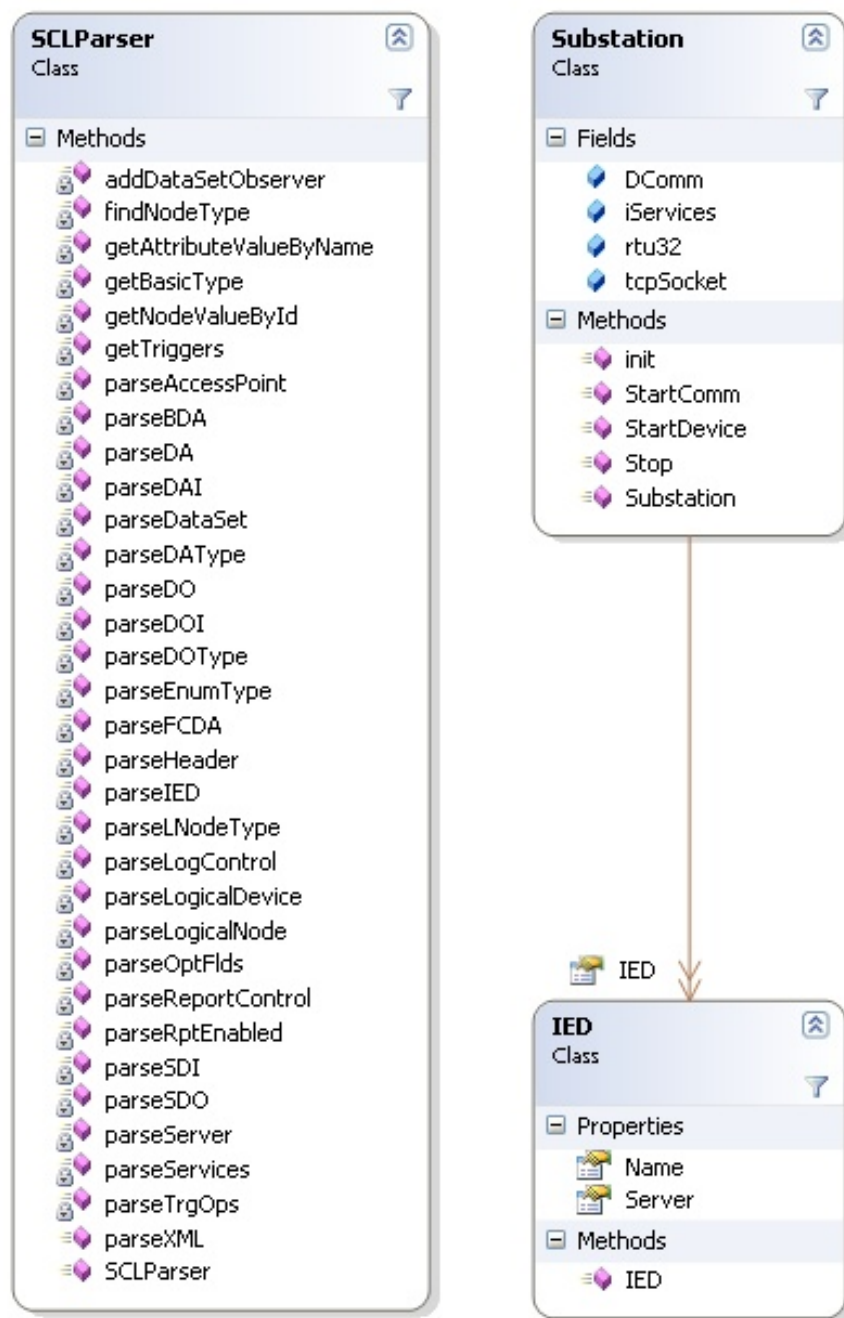


Figure 3.13: UML Class Diagram for the Substation module with attributes, properties and methods

communication module and thus "turns on" the server. The IED class represents the object which contains the server object of the data model.

## 3.7 SCL Configuration

According to the IEC61850 standard, SCL is the means of configuring a substation and describing such a configuration. SCL was introduced in 2.1.4. A full-scale commercial IEC61850 server should probably include a tool for editing the SCL configuration graphically. No such tool is part of this design, but an SCL parser is included. The SCL files parsed by this parser can be generated either manually, or by the use of a third-party SCL editor.

For this project, Kalki SCL Manager has been applied. Kalki SCL Manager is a graphical editor for configuring substations and is developed by Kalki Communication Technologies [17]. A trial version can be downloaded from Kalki's website, where the SCL Manager is described as "the leading Substation Configuration Software Platform, that enables implementation of complete configuration and engineering schemes as per IEC 61850." [17]

The proposed solution is not dependent on the Kalki SCL Manager in any way. This piece of software has only been used as a quick way of constructing SCL test files representing substations according to the rules laid out in the standard. The Kalki SCL Manager rejects files which do not comply with the IEC61850 standard. It should be mentioned that SCL files generated by Kalki SCL Manager are usually valid according to the Siemens sponsored SCD file validation on [1]. The SCL files used for testing in relation with this project have been validated according to the IEC61850 standard.

The Substation class is the main program and it invokes the SCLParser class (see figure 3.13) which is used to read and parse the validated SCL configuration file. The parsed content of the SCL file is used to generate a SERVER instance populated with logical devices, logical nodes etc. according to the SCL configuration file and in compliance with the IEC61850 data model.

## 3.8 Conclusion

In this chapter, the design of the solution has been presented. The solution has been divided into five modules, substation, information model, information exchange model, device model and communication. Each of these modules has been explained by use of UML class diagrams illustrating involved classes and their properties and methods. Also, some of the operations of each of the modules have been visualised by sequence diagrams.

The information model is the module containing the data model which consists of the server, logical devices, logical nodes, data and data attributes. The server has been designed to keep track of the content of the entire model by maintaining a list of object references. The information model can thereby handle basic IEC61850 service requests such as `GetServerDirectory` and `GetLogicalDeviceDirectory`.

The information exchange model is the module making other service requests available, such as reporting and logging. It has been designed with the control blocks necessary for buffered and unbuffered reporting as well as logging. Also the `DATASET` class has been designed which couples control blocks with data in the data model. Observer design pattern interfaces have been designed in suitable places to keep different objects updated based on changes of measured RTU inputs and user controlled changes.

The communication module has been designed to allow basic server/client connection and transmission of requests and responses. The communication is transmitted in binary form and is serialised by a compact formatter. The communication module is not designed according to the SCSMs of the IEC61850 standard.

The device model has been designed to provide input and output functionality to the RTU. The RTU's internal I/O database `wtool32` is used to continuously monitor changes in inputs. Data model entries are then correspondingly updated by use of the observer design pattern. Similarly, if values in the data model are manipulated, the RTU makes sure that any outputs which are connected to the changed values, are correspondingly changed.

The substation module has been designed as the main program which initialises the entire system based on the SCL configuration file.

Chapter 4 will present the implementation of the system.

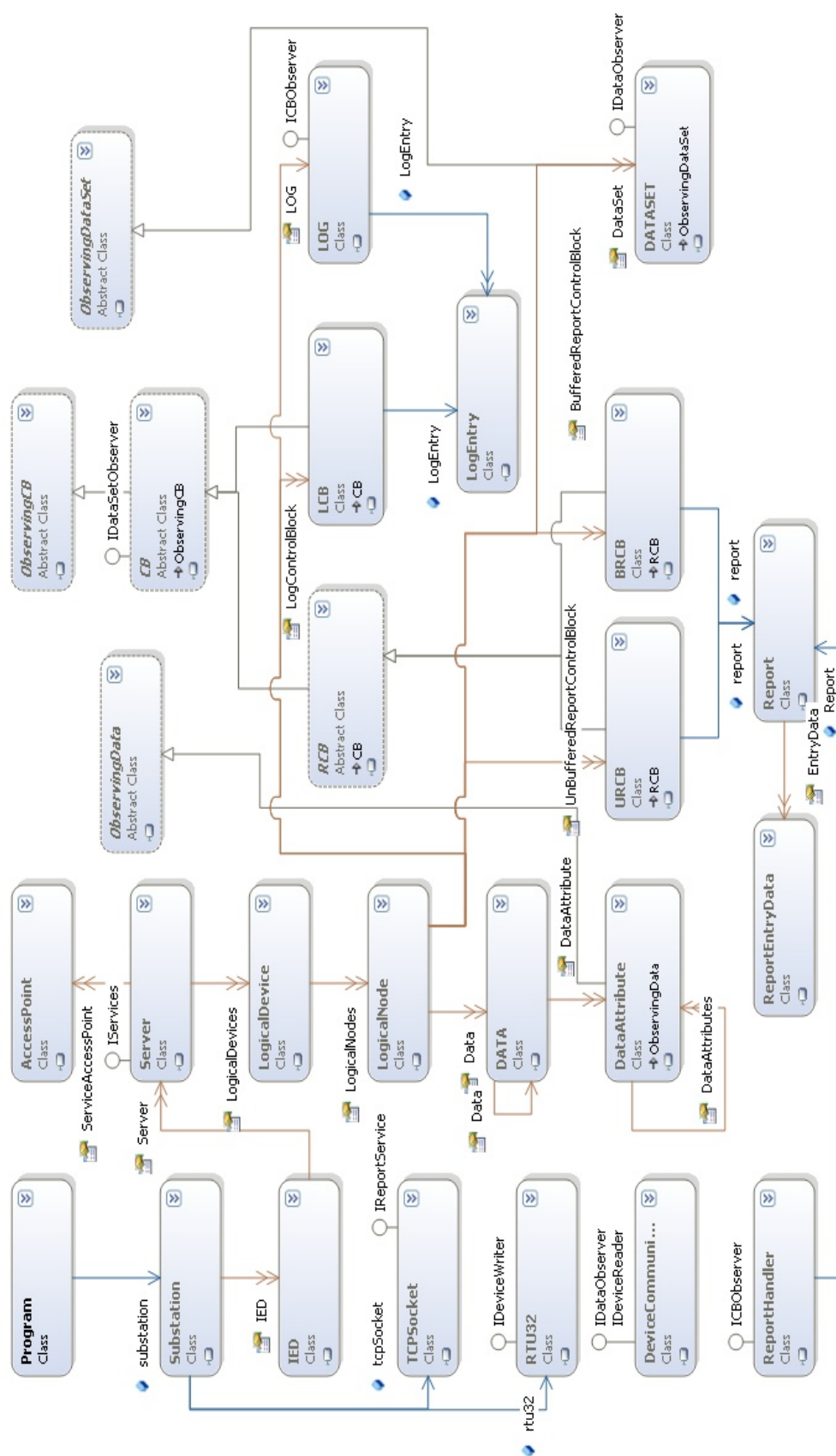


Figure 3.14: UML Class Diagram for the whole system

# IMPLEMENTATION

---

Based on the analysis and design chapters, this chapter will present the implementation of the IEC61850 server. The implementation of the system will be presented in the same order as in the design chapter. First, an overview of the general system is given in section 4.1. Then, the implementation of the information model will be explained in section 4.2. Section 4.3 describes the implementation of the information exchange model. The communication model is described in section 4.4. The implementation of the device model is presented in section 4.5 and the substation module which ties together the different modules, is described in section 4.6. Section 4.7 concludes the chapter.

## 4.1 Implementation of IEC61850 System

The implementation of the IEC61850 server shall be based on the design presented in chapter 3. The implementation process, therefore, is framed to a certain extent since many important choices have already been made during the design. Thus, the design of the system defines the system in broad outline, while the implementation consists of deciding in more detail *how* the design is turned into a real software solution.

The implementation of the system has been carried out with Microsoft Visual Studio 2005 as one solution. The solution contains projects for each of the modules as well as for other independent components used. The solution is contained in the file `IEC61850.CompliantSolution.sln`. The projects are added to the solution as class libraries, and in order for a class library to run under Windows CE, they are added by right-clicking the solution, selecting Add - New Project - Visual C# - Smart Device - Windows CE 5.0 and selecting Class Library. This adds "A project for creating a .NET Compact Framework 2.0 class library (.dll) for Windows CE 5.0 and later".

The complete solution contains the following 12 projects:

- **AuxFunc**  
Auxiliary functions for server/client communication and timer functionality

- **IEC61850.CompliantSolution**  
The main program for running the solution on the RTU under Windows CE
- **CompactFormatter**  
Functionality for serialising data on the compact framework
- **CompactObserver**  
Observer design patterns implemented on the compact framework
- **IEC61850.Communication**  
Functionality for creating a socket connection for client/server communication
- **IEC61850.Device**  
Library with functions for I/O to and from RTU
- **IEC61850.InformationExchangeModel**  
Library with classes of the information exchange model
- **IEC61850.InformationModel**  
Library with classes of the information model
- **IEC61850.Interfaces**  
Library with interfaces between modules of the solution
- **IEC61850.Substation**  
Library with the substation class which initialises the substation model
- **IEC61850.Types**  
Library with implementation of types specific for the IEC61850 server
- **IEC61850.UnitTests**  
Unit tests of the system

As classes, methods and properties are explained throughout this chapter, the following conventions shall be used to avoid misinterpretation and make the text concise.

- Class names are written in normal font, with capital first letter, for example `Server`
- Methods are written in typewriter font with arguments placed in parenthesis, but without their type
- Arguments of methods are written with small first letter if they are of simple types such as `string`, `int` and `bool`. When they are of IEC61850-specific types, they are given representative names and written in capital or capital first letter, for instance in the method

`newDataAttribute(DATA, name, FC, TrgOpList)`

DATA is of type `DATA`, name is of the simple type `string`, FC is of type `FunctionalConstraint` and TrgOpList is a list of type `TriggerConditions`

As it has been mentioned previously, the IEC61850 server implemented here is not a complete implementation fully compliant with the standard. However, it has been attempted to make an implementation with the features needed to satisfy the requirements of section 2.4. In some cases, properties are added which eventually are not used for the purpose intended in the standard. They have been included, nevertheless, in order to make the implementation generic and to make it a reasonable foundation for future work. In such cases, the intent of the standard shall be explained for such properties, though they may not be used for anything in this implementation.

## 4.2 Information Model

The implementation of the information model shall be based on the analysis of section 2.1.3 and the design of section 3.2. The information model consists of classes for the five elements of the data model, namely server, logical device, logical node, data and data attributes. As previously explained, the data model is hierarchical and the server is the topmost object in the model.

The Server class shall therefore contain the other classes in a hierarchical structure, as was also illustrated in the UML class diagram of figure 3.2. This is accomplished by declaring the Server object with a property `LogicalDevices` which is a list of logical devices. The class `LogicalDevice` will then contain a property for logical nodes and so on. This way, each of the objects of the data model are contained in an object directly above it in the hierarchical data model.

As the Server is the topmost object in the data model, it is practical to make additional functionality available to the Server class. For instance, the property `References` contains a sorted list of pairs of object references and strings which represent the referenced object. Recall from page 28 that an object reference has the format

```
LD/LN.Data[.Data[. ...]]DataAttribute[.DAComponent[. ...]]
```

The list of references is intended to make it possible, via the Server, to reference objects by their object reference. This way, all objects in the data model become accessible directly via the Server object. Objects may, however, be accessible in other ways too, and as will be shown later, an object is also reachable from its containing object, for instance a data object can be reached from the logical node object which contains it. The Server class also contains methods for adding a new reference to the list in the `References`-property and for removing a reference. These are called `addNewReference(objRef, Obj)` and `removeReference(objRef)`.

In addition to keeping track of all data model objects, the Server class has methods for creating the objects of the data model. Thus, the whole content of the data model



shall be initially created through method calls to methods in the Server class. The methods creating these objects are listed below:

```
newLogicalDevice(name, desc)
newLogicalNode(LD, lnClass, inst, prefix, desc)
newDATA(LN, name)
newCompositeDATA(DATA1, name)
newDataAttribute(DATA, name, FC, TrgOpList)
newDataAttributeComponent(ParentDATA, name)
```

Each of these methods work in a similar way. For example `newDATA(LN, name)` adds the name of the DATA object to the list of DATA contained in the logical node LN. Furthermore, a call is made to `addNewReference` which makes an entry into the list of object references in `Server.References`-property. This makes sure that the list of object references in the Server object is updated. As an example the code for the method `newLogicalNode()` is given below:

```
public LogicalNode newLogicalNode(
    LogicalDevice logicalDevice,
    String lnClass, String inst,
    String prefix, String desc)
{
    LogicalNode ln = new LogicalNode();
    ln.LNName.Value = lnClass + inst;
    ln.LNRef.Reference =
        logicalDevice.LDRef.ToString() + "/" +
        prefix + lnClass + inst;
    logicalDevice.LogicalNodes.Add(ln);
    addNewReference(ln.LNRef, ln);
    return ln;
}
```

The initialisation of the system is based on the content of the SCL configuration file on the RTU and this is explained in section 4.5. Furthermore, the data objects shall be updated whenever inputs on the RTU are scanned. This scan is accomplished outside the data model (see section 4.5), but the Server object contains the method used for the updating proces. This method is called `updateData()`.

In addition to the methods mentioned above, the Server class finally has three methods for adding data sets and control blocks for reporting and logging. These are added in exactly the same manner as the data model objects, but are always added to a logical node in compliance with the IEC61850 standard. The methods are defined as follows:

```
newDataSet(LN, name, desc, isDeletable)
newReportControlBlock(LN, name, datSet, intgPd, confRev, bufTime,
```

```

        buffered, rptID, desc, OptFldsList, TrgOp,
        enabled, ReportHandler)
newLogControlBlock(LN, name, datSet, logName)

```

As such, these objects are related to the information exchange model rather than the information model, but they shall be contained in logical nodes and are therefore created by the Server object and kept in the `References`-property along with object references of the data model.

The logical device, logical node, data and data attribute are defined in the classes `LogicalDevice`, `LogicalNode`, `DATA` and `DataAttribute`. Each may be seen as a container of objects directly beneath it in the data model. The containers are implemented as lists, accessible through the object's properties.

The logical node, however, is a special class in IEC61850 terms, and therefore, the `LogicalNode` class has a number of additional properties for the information exchange model, namely `DataSet`, `BufferedReportControlBlock`, `UnBufferedReportControlBlock`, `LogControlBlock` and `LOG`. The use of these is explained in section 4.3.

The `DATA` class contains a property `DataAttribute` which is a list of the contained data attributes. The `DATA` class is defined as a subclass of the `ObservingData` class of the `CompactObserver` library. The `ObservingData` class contains methods implementing the observer design pattern as described in section 3.3.4. The `ObservingData` class corresponds to the `Subject` class in the example in figure 3.8. It contains methods `Notify`, `Attach` and `Detach`.

These methods allow a concrete observer, in this case a `DATASET` object, to be attached to the data object, i.e. to subscribe to changes or updates in the particular data object. The `DATA` class corresponds to the `ConcreteSubject` in the figure 3.8 while the `IDataObserver` interface is exemplified by the `Observer` in the figure.

The `DATASET` (concrete observer) can also be detached, but as long as it is attached, the `DATA` object will notify the `DATASET` object of any data attribute value changes or updates, depending on the trigger conditions selected.

The idea of this whole construction is that a measurement of the inputs to the RTU shall immediately be written to the correct data object in the data model, and shall then propagate immediately to the `DATASET` which is used for reporting and logging. This allows a report to be transmitted and inform clients of the changes as they happen and without polling.

The DATASET correspondingly contains the inherited method `update` which updates the particular values which are changed.

In addition to the inherited methods for the observer design pattern, the DATA class contains the FCDA property, a sorted list of object references and corresponding functional constraints for those contained data attributes which have functional constraints. In relation to this, the DATA class has the following three methods for functional constraints:

- `addFCDA(objRef, FC)`  
adds a functionally constrained data attribute with the specified object reference and the specified functional constraint
- `isFunctionallyConstrained()` returns true if the DATA object has any functionally constrained data attributes
- `getFCDAWithFC(FC)` returns a list of object references of those data attributes which have the specified functional constraint

### 4.3 Information Exchange Model

It has already been explained how objects of the DATASET class are updated through the observer design pattern according to changes in the DATA objects of the data model. The DATASET is an important object in relation to the information exchange model in that it is used to tie a control block to the data that are to be reported, logged or otherwise used in service requests from a client.

The DATASET class is defined as a subclass of the `ObservingDataSet` class and is also defined to use the `IDataObserver` interface. This is done in order for the DATASET to be monitored by the control blocks, i.e. objects of the class CB (for Control Block) or its subclasses. Thus, the DATASET class appears in observer design patterns in two different ways, both as the observer, in relation to DATA, and as the subject, in relation to control blocks.

The DATASET has properties for name and object reference, like most other objects relevant to the IEC61850 standard. Furthermore, it has a property `isDeletable` of type boolean which is compliant with the standard, and indicates whether the DATASET may be deleted. It also has a string property `Desc` which may optionally contain a description of the DATASET.

The most important property of a DATASET object is `DSMemberRef` which is a list of object references contained in the DATASET. This list is initialised by the Server class

according to the SCL configuration file, and report control blocks and others monitor the values of this the objects in this list.

The control blocks for information exchange services are defined with the abstract class CB according to the standard. The CB class is defined as a subclass of class ObservingCB. This is again to allow for monitoring in an observer pattern, not as a demand of the standard. The CB class is monitored by the classes ReportHandler and LOG so that reports and logs are updated and ready to be transmitted as soon as changes take place in the data model and without the need of polling.

The CB class has properties for name, reference, optional fields, whether it is enabled or not, trigger conditions and a property for integrity period, indicating the time between periodical transmissions to ensure integrity of reports and logs. Furthermore, the CB class has a property `DataSet` which is the object reference of the DATASET monitored by the control block.

The information exchange model is extended further according to the standard with two subclasses of CB, namely LCB (Log Control Block) and RCB (Report Control Block) which again has the two subclasses BRCB (Buffered Report Control Block) and URCB (Unbuffered Report Control Block).

The RCB class is defined with properties `BufTm`, `ConfRev`, `Desc`, `GI`, `RptID` and `SqNum`.

- `BufTm` is intended to specify the amount of time for buffering of internal notifications caused by trigger conditions `dchg`, `qchg` and `dupd`. From the first notification a timer is set which expires after the amount of time specified by `BufTm`. In this period, all events are buffered and when the timer expires, the events are included in one report.
- `ConfRev` is the configuration revision number and represents the number of times the referenced DATASET has been edited.
- `Desc` contains an optional description. `GI` indicates whether General Interrogation is active.
- `RptID` is the unique identification of the report and is of type string. It is defined in the SCL file. `SqNum` is the sequence number and shall be incremented each time a report is sent.

BRCB and URCB are defined with the RCB class as abstract class, and inherit all properties and methods from the RCB class. In addition, BRCB has

- boolean property `PurgeBuf` indicating that buffered events that are not yet sent, shall be discarded

- `EntryID`-property of type `EntryID` used to identify an entry in a sequence of events in a buffered report
- `TimeOfEntry`-property of type `EntryTime` which indicates the time when the entry is added to the buffer

and `URCB` has

- the boolean property `Resv` indicating whether the control block is exclusively reserved by a client in which case that client is the only one who can set attributes of the `URCB`.

Finally, both the `BRCB` and `URCB` classes contain a method

```
notifyReport(DataReference, DATAAttribute, Trigger)
```

which creates an entry for a `Report` object. If the `Report` object is not created, the `notifyReport`-method creates it by a call to the `Report` class constructor. The entry is of type `ReportEntryData`, defined in the class with the same name. The `Report` object also has its own type and is defined in the `Report` class.

The `LCB` is defined as a subclass of `CB` and is used for creating logentries in the `LOG` object. It is a very simple class. It overrides the method

```
notifyLog(DataReference, DATAAttribute, Trigger)
```

which is defined in the `CB` class as part of the observer. Each time this function is called a new `LogEntry` is created and passed on to the `LOG` class.

The `LOG` class is used to store changes and updates to the data model for future retrieval - if e.g. the client wants to know the status of the substation at a given time. The `LOG` receives `LogEntries` from the `LCB` class and stores them in a buffer. This buffer is designed as a circular buffer such that after a number of entries the buffer begins to overwrite the old entries. The `LOG` class has two pairs of properties to keep track of this circular buffer. First there are timestamps for oldest and newest entry - `OldEntrTm` and `NewEntrTm` respectively. Second there are two integer pointers that point to the oldest and newest entry - `OldEntr` and `NewEntr`. These two pointers control the circular buffer - making sure that the new entries are entered at the right place in the buffer.

## 4.4 Communication Module

The communication module makes it possible for a client to connect and request services. The module consists of the `TCPSocket` class, the `ConnectionHelper` class and the `Envelope` class. The `TCPSocket` class creates two sockets, `listener` for listening

for client requests, and `socSender` for transmitting responses and server-initiated communication such as event reports.

The `Envelope` class defines the format used for transmitting data back and forth between the server and client. It contains a string property for the function and an array list property for the arguments.

The `TCPSocket` class contains the method `DoCommand` which is intended to handle requests by the client. The method takes an argument of type `Envelope` and executes the command issued by the function property with the arguments contained in the arguments property. The result is returned in a new `Envelope` object which is sent back to the client as response. If only values are sent back, they are contained in the arguments property and the function property is empty and shall be disregarded by the client.

The `ConnectionHelper` class contains the compact formatter mentioned in section 3.4. This formatter makes it possible to serialise data on the Compact Framework. For transmissions over a TCP socket, all data must be sent in binary form and is therefore serialised with the compact formatter.

## 4.5 Device Module

The device module is the part of the system that handles input and output to the RTU. It consists of the classes `RTU32`, `WTOOL32`, `SystemLog` and `DeviceCommunication`. These are all placed in the `IEC61850.Device` project.

The `WTOOL32` class is an interface to the `wtool32.dll` library which is provided in the RTU beforehand as the means of achieving input and output from and to the RTU. This is a DLL library of functions for writing to the internal `wtool32` database on the RTU and reading from it. This database is continuously scanning inputs on the RTU and contains the values of these inputs. Similarly, output is made by writing values to this database upon which the output ports are set accordingly by the RTU. The functions are made available in the IEC61850 server by DLL import.

The `RTU32` class contains a thread `TRtu` which runs continuously. When it starts, it initialises the `wtool32` database by calling the `WTool32Init()` -method of the `wtool32` class. Thus, the thread is at all times scanning the inputs of the RTU. The values of the inputs are sent to the `DeviceCommunication` class through the `IDeviceReader` interface. It updates the data model accordingly.

The `DeviceCommunication` class updates data values through the `updateValue`-method. Furthermore, it observes the data model and is notified when changes occur in data attributes. This may happen when the client issues control services assigning values to data attributes. When such changes occur, it shall update the RTU in the other direction, through the `IDeviceWriter` interface.

## 4.6 Substation Module

The Substation module is the main module for the implementation. At startup the main method in `Program.cs` creates a new instance of a substation. The Substation constructor has a path to an SCL file as argument.

The constructor creates first an instance of a `TCPSocket` that is used later for communication with the client. Secondly an `SCLParser` is created which parses through the SCL file and thus creates the Information Model and Information Exchange Model for the whole substation.

The `SCLParser` iterates through the SCL-file by first creating a instance of an `XmlDocument` class. This instance provides functionality to navigate through the entire file by starting at the `RootElement`. Each `XmlNode` element typically has some `childNodes` and some attributes. In order to parse through the entire file the parser must make sure that every `childNodes` of an `XmlNode` is visited. To configure a `XmlNode` its collection of attributes are parsed one by one. These attribute names are known beforehand and these are only parsed in order to get their assigned values. Most of the parse-methods are quite similar to this function signature:

```
parseLogicalNode(Server, LogicalDevice, XmlNode)
```

The `Server` is used to pair all object references with the corresponding object for later retrieval. The `LogicalDevice` is the "parent" of this `LogicalNode`, so this `LogicalNode` is stored in the list of `LogicalNodes` contained in the `LogicalDevice`. The final argument is the current `XmlNode` that shall be parsed. The code for parsing a logical node is given below:

```
parseLogicalNode(Server server, LogicalDevice
logicalDevice, XmlNode node)
{
    String lnClass = getNodeValueById(node, "lnClass");
    String lnType = getNodeValueById(node, "lnType");
    String inst = getNodeValueById(node, "inst");
    String prefix = getNodeValueById(node, "prefix");
    String desc = getNodeValueById(node, "desc");
```

---

```

LogicalNode logicalNode = server.newLogicalNode(
    logicalDevice,
    lnClass,
    inst,
    prefix,
    desc);

//Parse children of node
XmlAttributeCollection atc = node.Attributes;
XmlNode n = findNodeType("LNodeType", "id", lnType);
parseLNodeType(server, logicalNode, n);
foreach (XmlNode xn in node.ChildNodes)
{
    switch (xn.Name)
    {
        case "DOI":
            parseDOI(xn);
            break;
        case "DataSet":
            parseDataSet(server, logicalNode, xn);
            break;
        case "ReportControl":
            parseReportControl(server, logicalNode, xn);
            break;
        case "LogControl":
            parseLogControl(server, logicalNode, xn);
            break;
    }
}
}

```

The first part of the method defines the instance of LogicalNode. In the second part of the method the parser iterates through the childNodes of this XmlNode. These are DOI (Instantiated Data Object), DataSet, ReportControl and LogControl instances.

Creating the Information and Information Exchange Model consists of creating IED, SERVER, LogicalDevice, LogicalNode, DATA and DataAttribute instances, adding observers to the DATA, creating DATASET and ControlBlocks. The SCL file is typically a large file, so the parsing can take a while to complete.

The constructor then initialises the Device Module, which starts producing inputs to the datamodel. Finally the communication module is started so that a client can connect to the server to receive reports or utilize the offered services. The system is now running and the RTU produces input to the data model which by using the



observer pattern can generate reports that can be sent to a connected client.

## **4.7 Conclusion**

This chapter has explained the implementation of the basic IEC61850 server. The overall structure of the solution is explained in section 4.1 where an overview is given of the projects used. Section 4.2 explains the implementation of the information model which contains the hierarchical data model with the basic objects of the standard. In section 4.3, the implementation of the information exchange model is described, where reporting and logging are implemented. Section 4.5 explains the implementation of the device module with RTU input/output capabilities. Section 4.4 explains the communication module which allows a client to request services from the server based serialised data transmission over a simple TCP socket connection. The implementation of the substation module is explained in section 4.6. It initialises the system based on the content of the SCL configuration file which is parsed. The implementation has been carried out based on the analysis and design of the standard. The server has been implemented on the Brodersen RTU32 and runs under the Windows CE operating system.

# TEST

---

In this chapter, the tests completed for the solution will be presented. First, a unit test is presented in section 5.1 which tests the parsing of the SCL configuration file and the contents of the data model. Second, a number of test cases are presented which have been tested using black box testing. These tests are presented in section 5.2. Section 5.3 concludes the test chapter.

## 5.1 Unit Test

A unit test has been created which tests whether the system contains the expected data model given a specific SCL file. The SCL file used for this test is included in Appendix C. The unit test project file is found on the CD. The services tested are GetServerDirectory, GetLogicalDeviceDirectory and GetLogicalNodeDirectory.

GetServerDirectory is invoked with parameter LogicalDevice. Expected outcome is a list of object references of all logical devices of the server. The outcome is as expected.

GetLogicalDeviceDirectory is invoked with the object reference of a logical device as parameter. Expected outcome is a list containing the logical nodes of the logical device. The outcome is as expected.

GetLogicalNodeDirectory is invoked with the object reference of a logical node and an ACSI class as parameters. Expected outcome is a list of instance names of all objects of the specified class in the logical node. The outcome is as expected.

## 5.2 Test Cases

This section presents cases which have been tested with black box testing. First, retrieval of the contents of the data model is tested. This consists of services such

as GetDataDirectory and GetDataSetDirectory. Second, the reporting and logging services are tested.

### 5.2.1 GetDataDirectory

**Test:** GetDataDirectory

**Execution:** The GetDataDirectory service is invoked with the object reference of a Data as parameter.

**Expected outcome:** The data attribute names of all data attribute contained in the specified Data are returned in a list.

**Observed outcome:** As expected.

### 5.2.2 GetDataSetDirectory

**Test:** GetDataSetDirectory

**Execution:** The GetDataSetDirectory service is invoked with the object reference of a dataset as parameter.

**Expected outcome:** The object references of all data set members contained in the specified dataset are returned in a list.

**Observed outcome:** As expected.

### 5.2.3 Logging

**Test:** Logging

**Execution:** An event is triggered (data updated, data changed or quality changed) in data attributes referenced by a data set which is related to a log control block.

**Expected outcome:** A log entry is made.

**Observed outcome:** As expected.

### 5.2.4 Reporting

**Test:** Report

**Execution:** An event is triggered (data updated, data changed or quality changed) in data attributes referenced by a data set which is related to a report control block.

**Expected outcome:** A report is generated. Depending on whether the control block is BRCB or URCB, the report shall be buffered or unbuffered, respectively.

**Observed outcome:** As expected.

### 5.2.5 Connecting to the Server

**Test:** Connecting to the Server

**Execution:** A test client connects to the server and requests the GetServerDirectory service with LogicalDevice as parameter.

**Expected outcome:** A list of logical devices on the server is sent to the client

**Observed outcome:** As expected.

## 5.3 Conclusion

This chapter has presented a unit test of the parsing of an SCL configuration file and of the resulting content of the data model. Also, black box tests of basic information model services have been performed and finally reporting and logging services and client/server communication have been black box tested. The outcomes of these tests are as expected.

# CONCLUSION

---

This chapter shall conclude the report. In section 6.1, a short summary is given of the conclusions of the analysis, design, implementation and tests performed. Then a summary of contributions is presented in section 6.2 and finally a list of suggestions for future work is presented in section 6.3.

## 6.1 Summary of Results

The introduction of chapter 1 gives an introduction to the area of the IEC61850 standard and places the standard in a historical context. An introduction is also given to Brodersen Controls A/S and the RTU32 provided by Brodersen. The chapter finally gives an overall problem statement indicating the following goals for the project:

- provide an overview and analysis of the IEC61850 standard
- perform an analysis of the scenarios envisioned by Brodersen and place them in relation to the scope of the IEC61850 standard
- design and implement a basic IEC61850 server

The analysis of chapter 2 outlines basic concepts of the IEC61850 standard and describes the different models defined in the standard. The hierarchical data model of the information model is described and the definitions of the main classes, Server, LogicalDevice, LogicalNode, DATA and DataAttribute are explained based on the definitions of [9], [10] and [11]. The logical node class definition is thoroughly explained; examples are given and logical nodes are grouped according to their functionality as prescribed in [11].

The Abstract Communication Service Interface - defined in [9] - which contains the services available for clients, is also explained in some detail. Basic information services are described which allow the client to retrieve the structure and objects of the data model, and other service models are described, such as services for reporting, logging, control, setting groups and so on.

The communication model defined in [12], [13] and [14] is described in section 2.1.8.

The scenarios provided by Brodersen Controls A/S are listed and analysed in sections 2.2. The cases are placed in relation to the scope of the IEC61850 standard, and requirements for IEC61850 classes and services are identified for an envisioned implementation of these scenarios. Some of these requirements, however, fall outside the scope of the standard.

The Brodersen RTU32 is briefly analysed in section 2.3 and it is explained how the Windows CE operating system and Compact .NET Framework can affect implementation.

Based on the analysis of the standard, the scenarios and the RTU, as well as on the problem definition, a specification of requirements is presented in section 2.4. The requirements are aimed at the implementation of an IEC61850 server with basic functionality such as reporting and logging.

Based on the analysis of chapter 2 and in particular the specification of requirements, chapter 3 outlines the overall architecture of the solution and explains the design of classes and how they interact. The solution is divided into five modules, the information model, information exchange module, communication module, device module and substation module. Each of the modules are described by use of UML class diagrams and some of the basic operations of classes of each module are visualised in sequence diagrams. It is explained how an observer design pattern can allow the information exchange model to be updated when changes happen to the data model, either because of input from the RTU or because of changes initiated from the client, without the use of polling.

In continuation of the design, chapter 4 explains the implementation of the IEC61850 basic server. The modules information model, information exchange model, device, communication and substation are explained. The substation module initialises the system by invoking the SCLParser which parses the SCL configuration file and generates a data model on basis of the SCL file. The information exchange module makes services available to a client. The device module supplies I/O capabilities to the RTU and the communication module makes it possible for a client to connect to the server and request services. The observer design pattern is implemented to keep the data model updated according to inputs and also to update control blocks which monitor the data model.

Chapter 5 gives an outline of the tests of the system and test results. Unit tests have been performed on static services while black box testing has been performed on services whose response are changing with time according to events. Outcomes are as

expected.

## 6.2 Summary of Contributions

The analysis of chapter 2 has given an overview of the IEC61850 standard and placed the Brodersen RTU32 in relation to the scope of the standard. This analysis should be a reasonable starting point for the company in planning how to place the RTU32 as a central component in the new communication structure implied in the standard.

In particular, the scenarios provided by Brodersen Controls A/S have been analysed with respect to the IEC61850 standard and a number of features have been identified which are outside the scope of the standard. These features however, could quite easily be implemented on the RTU, but would have to be carefully combined with the IEC61850 server.

In addition to providing this analysis, a basic IEC61850 server has been designed and implemented for the Brodersen RTU32. The following requirements were specified in section 2.4:

- The solution shall implement a basic IEC61850 server which is able to run on a Brodersen RTU32 under Windows CE
- It shall be possible to read a configuration file in SCL format for a substation and have a data model with a server instance generated on the basis of the SCL file
- A client shall be able to connect to the server and request basic ACSI services such as reporting and logging
- The solution should be generic so that extension is possible later

As the chapters 3, 4 and 5 have demonstrated, these requirements have been fulfilled. This should be a reasonably useful platform for further development of a fully compliant IEC61850 system for the RTU32, since it has been strived to make the implementation generic and easy to extend. The comprehensive nature of the IEC61850 standard means that there are rich possibilities for enhancing the implementation. Some suggestions for enhancements are given in section 6.3 which follows.

## 6.3 Discussion and Future Work

This section lists suggestions for future work on the IEC61850 basic server. The suggestions are divided into improvements of the implemented system (6.3.1) and

suggested additional IEC61850 functionality (6.3.2). In relation to this, it can be mentioned that the supervisors and authors contemplate preparing an article on the work carried out in this thesis in continuation of work by Kostic et al [16] and Schwarz [20].

### **6.3.1 Improvements to the Basic IEC61850 Server Implementation**

- **More detailed data model:** The data model designed and implemented consists of only one class for logical node. Instead, all 91 logical node classes could be designed as individual classes, which would make it possible to incorporate the SCL check of the generated file in the constructor of each class. This would also make it possible to ensure that rules regarding mandatory and optional attributes are fulfilled at all times. Alternatively, a procedure should be implemented which could ensure that all logical nodes in the data model at all times comply with the rules for mandatory and optional attributes.
- **Prioritised alarms:** Functionality for giving higher priorities to particular event reports could be added to the system. This would mean that the report handler should take into account the priority of reports, and possibly suppress reports with lower priority until reports with higher priority were sent. The client should be able to set and change the priority of datasets.
- **Implement Brodersen scenarios:** The scenarios provided by Brodersen are analysed in section 2.2 where it is uncovered that some extra functionality outside the IEC61850 scope is needed. This includes functionality for mapping for instance two input ports to one data model element. The mapping could be made with some logical operators such as XOR. This would be a suitable extension to the basic IEC61850 server implemented.
- **SCL Editor:** For this project, the Kalki SCL Manager has been used to generate SCL files for testing. In a full scale IEC61850 implementation, an IEC61850-compliant SCL editor with file validation would be a useful feature for graphically configuring a substation.

### **6.3.2 Additional IEC61850 Functionality**

- **Association:** The connection between a client and the IEC61850 server in the solution presented in this project is a simple procedure which does not take into account the Association service where clients receive an association ID upon connecting to the system. The Association service must be implemented in order for the system to keep track of which clients have received which reports and alarms, and control which clients are allowed to edit which data sets and so on.



- **Multiple Clients:** The system can currently handle only one client, and in order to extend it to handle multiple clients, concurrency issues must be considered as well as proper access control. Also, as mentioned above, the association service must be fully implemented.
- **Full ACSI:** The proposed system is a basic IEC61850 server and there are a number of services which have been disregarded in the design and implementation. Association has been mentioned, but services such as those of the control model and setting group are also important for such a server to be of practical use. The client should then be able to make use of the Select before operate services in the control model. The system could undoubtedly be extended to comprise a full ACSI, but for some services, such as GOOSE, the SCSM of the standard using MMS over Ethernet would need to be implemented.
- **SCSM:** The system has been implemented with only a simple socket connection between the client and server. The IEC61850 defines that the MMS protocol shall be included in the SCSM along with Ethernet. Possibly, web services will become part of the standard in the future and would also be a possibility. The communication module of the system could be replaced by one supporting MMS over Ethernet and the other requirements of the SCSMs. This would make the system interoperable which it is not currently.

## BIBLIOGRAPHY

---

- [1] Siemens AG. SCL validator and webpage, sponsored by UCA and Siemens. <http://www.61850.com/>. A note on this webpage: The SCL validator itself is available on the index page, but other resources cannot be reached from it. Menu appears on e.g. <http://www.61850.com/introduction.html> which must be entered manually.
- [2] Klaus Peter Brand, Volker Lohmann, and Wolfgang Wimmer. *Substation Automation Handbook*. Utility Automation Consulting Lohmann, 2003.
- [3] Brodersen Controls A/S. Brodersen Controls A/S Website. <http://www.brodersencontrols.dk/>.
- [4] Brodersen Controls A/S. RTU32 - A Universal Controller (Brochure).
- [5] CET Center for Elteknologi. Projects at CET. <http://www.dtu.dk/centre/cet/forskning/projekter.aspx>.
- [6] IEC. IEC61850-5: Communication requirements for function and device models.
- [7] IEC. IEC61850-6: Substation automation system configuration description language.
- [8] IEC. IEC61850-7-1: Basic communication structure for substation and feeder equipment - Principles and models.
- [9] IEC. IEC61850-7-2: Basic communication structure for substation and feeder equipment - Abstract communication service interface (ACSI).
- [10] IEC. IEC61850-7-3: Basic communication structure for substation and feeder equipment - Common data classes.
- [11] IEC. IEC61850-7-4: Basic communication structure for substation and feeder equipment - Compatible logical node classes and data classes.
- [12] IEC. IEC61850-8-1: Specific communication service mapping (SCSM) - Mapping to MMS (ISO/IEC 9506 Part 1 and Part 2).
- [13] IEC. IEC61850-9-1: Specific communication service mapping (SCSM) - Serial unidirectional multidrop point to point link.

- [14] IEC. IEC61850-9-2: Specific communication service mapping (SCSM) - Mapping on a IEEE 802.3 based process.
- [15] Hubert Kirrmann. Introduction to IEC 61850 substation communication standard. *ABB Switzerland Ltd, Corporate Research, ABBCH-RD*, 2004.
- [16] T. Kostic, C. Frei, O. Preiss, and M. Kezunovic. Scenarios for data exchange using standards IEC 61970 and IEC 61850. *UCA User Group meeting, Cigre Paris Session 2004*, pages 3–4, 2004.
- [17] KALKI Communication Technologies Limited. Kalki communication technologies webpage. <http://www.kalkitech.com/>.
- [18] R. E. Mackiewicz. Overview of IEC61850 and Benefits. *Power Systems Conference and Exposition, 2006. PSCE '06. 2006 IEEE PES*, pages 623–630, 2006.
- [19] Microsoft. Compact framework 2.0 webpage at microsoft developers network. [http://msdn2.microsoft.com/en-us/library/2weec7k5\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/2weec7k5(VS.80).aspx).
- [20] Karlheinz Schwarz. IEC 61850, IEC 61400-25 and IEC 61970: Information models and information exchange for electric power systems. *Distributech*, 2004.
- [21] Angelo SCotto. CompactFormatter Website. <http://www.freewebs.com/compactFormatter/About.html>.

# GLOSSARY

---

AA - Application Association  
ACSI - Abstract Communication Services Interface  
API - Application Program Interface  
ASDU - Application Service Data Unit  
ASN.1 - Abstract Syntax Notation One  
BDA - Basic Data Attribute (that is not structured)  
BRCB - Buffered Report Control Block  
CDC - Common Data Class (IEC 61850-7-3)  
CIM - Common Information Model for energy management applications  
CT - Current Transducer  
DA - Data Attribute  
DAI - Instantiated Data Attribute  
DataRef - Data Reference  
DAType - Data Attribute Type  
dchg - data change trigger option  
DO - DATA in IEC 61850, data object type or instance, depending on the context  
DOI - Instantiated Data Object (DATA)  
DS - Data Set  
DTD - Document Type Definition  
dupd - data-update trigger option  
DUT - Device Under Test  
FAT - Factory Acceptance Test  
FC - Functional Constraint  
FCD - Functionally Constrained Data  
FCDA - Functionally Constrained Data Attribute  
GI - General Interrogation  
GoCB - GOOSE Control Block  
GOOSE - Generic Object Oriented Substation Event  
GPS - Global Positioning System (time source)  
GsCB - GSSE Control Block  
GSE - Generic Substation Event  
GSSE - Generic Substation Status Event  
HMI - Human Machine Interface  
I/O - Input and Output contact or channels (depending on context)

ICD - IED Capability Description  
ID - IDentifier  
IED - Intelligent Electronic Device  
IF - (Serial) Interface  
IntgPd - Integrity Period  
IP - Internet Protocol  
LAN - Local Area Network  
LC - Logical Connection  
LCB - Log Control Block  
LD - Logical Device  
LDInst - Instantiated Logical Device  
LLN0 - Logical Node Zero  
LN - Logical Node  
LNinst - Instantiated Logical Node  
LPHD - Logical Node Physical Device  
MC - MultiCast  
MCAA - MultiCast Application Association  
MICS - Model Implementation Conformance Statement  
MMS - Manufacturing Message Specification (ISO 9506 series)  
MSV - Multicast Sampled Value  
MSVCB - MultiCast Sampled Value Control Block  
MsvID - ID for MSV  
NCC - Network Control Center  
OSI - Open System Interconnection  
PC - Physical Connection  
PD - Physical Device  
PDIS - DIStance Protection  
PDU - Protocol Data Unit  
PHD - Physical Device  
PICOM - PIece of COMmunication  
PICS - Protocol Implementation Conformance Statement  
PIXIT - Protocol Implementation eXtra InformaTion  
PTOC - Time OverCurrent Protection  
PTRC - TRip Conditioning  
qchg - Quality Change Trigger Option  
RTU - Remote Terminal Unit  
SAS - Substation Automation System  
SAT - Site Acceptance Test  
SAV - Sampled Analogue Value (IEC 61850-9 series)  
SBO - Select Before Operate  
SCADA - Supervisory Control And Data Acquisition  
SCD - Substation Configuration Description  
SCL - Substation Configuration Language  
SCSM - Specific Communication Service Mapping

SDI - Instantiated Sub Data; middle name part of a structured DATA name  
SG - Setting Group  
SGCM - Setting Group Control Block  
SMV - Sampled Measured Value  
SoE - Sequence-of-Events  
SSD - System Specification Description  
SUT - Service Under Test  
SV - Sampled Values  
SVC - Sampled Values Control  
TCI - TeleControl Interface (e.g. to NCC)  
TCP - Transport Control Protocol  
TMI - TeleMonitoring Interface (e.g. to engineers workplace)  
TP - Two-Party  
TPAA - Two Party Application Association  
TrgOp - Trigger Option  
UCA - Utility Communication Architecture  
UML - Unified Modelling Language  
URCB - Unbuffered Report Control Block  
URI - Universal Resource Identifier  
USVCB - Unicast Sampled Value Control Block  
UsvID - ID for USV  
UTC - Coordinated Universal Time  
VMD - Virtual Manufacturing Device  
VT - Voltage Transducer  
XCBR - Circuit BReaker  
XML - eXtensible Markup Language

# DETAILS OF THE IEC61850 STANDARD

---

## **B.1 List of Logical Node Groups**

Table B.1.1 shows the groups of logical nodes defined in 5.1 in [11].

## **B.2 ACSI Classes and Their Services**

Table B.2.1 shows the complete list of ACSI classes and their services defined in 5.4 in [9].

---

Group Indicator	Logical Node Group
A	Automatic Control
C	Supervisory Control
G	Generic Function References
I	Interfacing and Archiving
L	System Logical Nodes
M	Metering and Measurement
P	Protection Functions
R	Protection Related Functions
S	Sensors, Monitoring
T	Instrument Transformer
X	Switchgear
Y	Power Transformer and Related Functions
Z	Further (Power System) Equipment

Table B.1.1: List of Logical Node Groups



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

Table B.2.1: Complete List of ACSI Classes and Their Services

Table B.2.2

# SCL TEST FILE

## SCL file used for unit test

```

1 <?xml version="1.0" encoding="UTF-16" ?>
2 - <!-- This file is generated using KALKI SCL Manager IEC61850 Configuration Tool (www.kalkitech.
   com)
3 ->
4 - <SCL xmlns="http://www.iec.ch/61850/2003/SCL">
5   <Header id="" version="" revision="" toolID="" nameStructure="IEDName" />
6   <IED name="IED1">
7     <Services>
8       <DynAssociation />
9       <GetDirectory />
10      <GetDataObjectDefinition />
11      <GetDataSetValue />
12      <DataSetDirectory />
13      <ConfDataSet max="0" maxAttributes="0" modify="false" />
14      <ReadWrite />
15      <ConfReportControl max="0" />
16      <GetCBValues />
17      <ConfLogControl max="0" />
18      <ConfLNs fixPrefix="true" fixLnInst="true" />
19      <GOOSE max="0" />
20      <FileHandling />
21    </Services>
22    <AccessPoint name="AccPoint1" desc="">
23      <Server>
24        <Authentication />
25        <LDevice inst="RTU" desc="">
26          <LN0 lnClass="LLN0" lnType="LLN01" inst="" desc="">
27            <DOI name="Mod">
28              <DAI name="ctlModel">
29                <Val>status-only</Val>
30              </DAI>
31            </DOI>
32          </LN0>
33          <LN lnClass="LPHD" lnType="LPHD1" inst="1" prefix="" desc="" />
34          <LN lnClass="CALH" lnType="CALH1" inst="1" prefix="" desc="">
35            <DOI name="Mod">
36              <DAI name="ctlModel">
37                <Val>status-only</Val>
38              </DAI>
39            </DOI>
40          </LN>
41          <LN lnClass="XSWI" lnType="XSWI1" inst="1" prefix="" desc="">
42            <DOI name="Mod">
43              <DAI name="ctlModel">
44                <Val>status-only</Val>
45              </DAI>

```

```

46 </DOI>
47 - <DOI name="Pos">
48 - <DAI name="ctlModel">
49 <Val>status-only</Val>
50 </DAI>
51 </DOI>
52 - <DOI name="BlkOpn">
53 - <DAI name="ctlModel">
54 <Val>status-only</Val>
55 </DAI>
56 </DOI>
57 - <DOI name="BlkCls">
58 - <DAI name="ctlModel">
59 <Val>status-only</Val>
60 </DAI>
61 </DOI>
62 </LN>
63 - <LN lnClass="MMXN" lnType="MMXNI" inst="1" prefix="" desc="">
64 - <DOI name="Mod">
65 - <DAI name="ctlModel">
66 <Val>status-only</Val>
67 </DAI>
68 </DOI>
69 </LN>
70 - <LN lnClass="PTUV" lnType="PTUV1" inst="1" prefix="" desc="">
71 - <DataSet name="Monitoring" desc="">
72 <FCDA ldInst="RTU" prefix="" lnClass="PTUV" lnInst="1" doName="Op" daName="general" fc="ST" />
73 <FCDA ldInst="RTU" prefix="" lnClass="CALH" lnInst="1" doName="GrAlm" daName="stVal" fc="ST" />
74 <FCDA ldInst="RTU" prefix="" lnClass="MMXN" lnInst="1" doName="Vol" daName="mag" fc="MX" />
75 <FCDA ldInst="RTU" prefix="" lnClass="XSWI" lnInst="1" doName="Pos" daName="stVal" fc="ST" />
76 </DataSet>
77 - <ReportControl name="myReport" datSet="Monitoring" intgPd="0" confRev="0" bufTime="0" buffered=
    "true" rptID="RTU" desc="">
78 <TrgOps dchg="true" dupd="true" />
79 <OptFields timeStamp="true" dataSet="true" reasonCode="true" />
80 </ReportControl>
81 - <LogControl name="Log" datSet="Monitoring" logName="rtuLOG">
82 <TrgOps dchg="true" qchg="true" />
83 </LogControl>
84 - <DOI name="Mod">
85 - <DAI name="ctlModel">
86 <Val>status-only</Val>
87 </DAI>
88 </DOI>
89 - <DOI name="StrVal">
90 - <SDI name="minVal">
91 - <DAI name="f">
92 <Val>50</Val>
93 </DAI>
94 </SDI>
95 </DOI>
96 </LN>
97 </LDevice>
98 </Server>
99 </AccessPoint>
100 </IED>
101 - <DataTypeTemplates>
102 - <LNNodeType id="LLN01" lnClass="LLN0">
103 <DO name="Mod" type="INC_1_Mod" />
104 <DO name="Beh" type="INS_1_Beh" />
105 <DO name="Health" type="INS_1_Beh" />
106 <DO name="NamPlt" type="LPL_1_NamPlt" />

```

```

107 </LNodeType>
108 - <LNodeType id="LPHD1" lnClass="LPHD">
109   <DO name="PhyNam" type="DPL_1_PhyNam" />
110   <DO name="PhyHealth" type="INS_1_Beh" />
111   <DO name="Proxy" type="SPS_1_Proxy" />
112 </LNodeType>
113 - <LNodeType id="PTUV1" lnClass="PTUV">
114   <DO name="Mod" type="INC_3_Mod" />
115   <DO name="Beh" type="INS_3_Beh" />
116   <DO name="Health" type="INS_3_Beh" />
117   <DO name="NamPlt" type="LPL_4_NamPlt" />
118   <DO name="Str" type="ACD_2_Str" />
119   <DO name="Op" type="ACT_2_Op" />
120   <DO name="StrVal" type="ASG_2_StrVal" />
121 </LNodeType>
122 - <LNodeType id="CALH1" lnClass="CALH">
123   <DO name="Mod" type="INC_1_Mod" />
124   <DO name="Beh" type="INS_1_Beh" />
125   <DO name="Health" type="INS_1_Beh" />
126   <DO name="NamPlt" type="LPL_2_NamPlt" />
127   <DO name="GrAlm" type="SPS_1_Proxy" />
128 </LNodeType>
129 - <LNodeType id="XSWI1" lnClass="XSWI">
130   <DO name="Mod" type="INC_1_Mod" />
131   <DO name="Beh" type="INS_1_Beh" />
132   <DO name="Health" type="INS_1_Beh" />
133   <DO name="NamPlt" type="LPL_2_NamPlt" />
134   <DO name="Loc" type="SPS_1_Proxy" />
135   <DO name="OpCnt" type="INS_1_Beh" />
136   <DO name="Pos" type="DPC_1_Pos" />
137   <DO name="BlkOpn" type="SPC_1_BlkOpn" />
138   <DO name="BlkCls" type="SPC_1_BlkOpn" />
139   <DO name="SwTyp" type="INS_1_Beh" />
140   <DO name="SwOpCap" type="INS_1_Beh" />
141 </LNodeType>
142 - <LNodeType id="MMXN1" lnClass="MMXN">
143   <DO name="Mod" type="INC_2_Mod" />
144   <DO name="Beh" type="INS_2_Beh" />
145   <DO name="Health" type="INS_2_Beh" />
146   <DO name="NamPlt" type="LPL_3_NamPlt" />
147   <DO name="Vol" type="MV_1_Vol" />
148 </LNodeType>
149 - <DOType id="LPL_1_NamPlt" cdc="LPL">
150   <DA name="vendor" bType="VisString255" fc="DC" />
151   <DA name="swRev" bType="VisString255" fc="DC" />
152   <DA name="d" bType="VisString255" fc="DC" />
153   <DA name="configRev" bType="VisString255" fc="DC" />
154   <DA name="ldNs" bType="VisString255" fc="EX" />
155 </DOType>
156 - <DOType id="INS_1_Beh" cdc="INS">
157   <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
158   <DA name="q" bType="Quality" fc="ST" qchg="true" />
159   <DA name="t" bType="Timestamp" fc="ST" />
160   <DA name="subEna" bType="BOOLEAN" fc="SV" />
161   <DA name="subVal" bType="INT8" fc="SV" />
162   <DA name="subQ" bType="Quality" fc="SV" />
163   <DA name="subID" bType="VisString64" fc="SV" />
164 </DOType>
165 - <DOType id="INC_1_Mod" cdc="INC">
166   <DA name="Cancel" type="INCCancel_1" bType="Struct" fc="CO" />
167   <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
168   <DA name="q" bType="Quality" fc="ST" qchg="true" />

```

```

169 <DA name="t" bType="Timestamp" fc="ST" />
170 <DA name="subEna" bType="BOOLEAN" fc="SV" />
171 <DA name="subVal" bType="INT8" fc="SV" />
172 <DA name="subQ" bType="Quality" fc="SV" />
173 <DA name="subID" bType="VisString64" fc="SV" />
174 <DA name="ctlModel" type="ctlModelEnum" bType="Enum" fc="CF" />
175 </DOType>
176 - <DOType id="SPS_1_Proxy" cdc="SPS">
177   <DA name="stVal" bType="BOOLEAN" fc="ST" dchg="true" />
178   <DA name="q" bType="Quality" fc="ST" qchg="true" />
179   <DA name="t" bType="Timestamp" fc="ST" />
180   <DA name="subEna" bType="BOOLEAN" fc="SV" />
181   <DA name="subVal" bType="BOOLEAN" fc="SV" />
182   <DA name="subQ" bType="Quality" fc="SV" />
183   <DA name="subID" bType="VisString64" fc="SV" />
184 </DOType>
185 - <DOType id="DPL_1_PhyNam" cdc="DPL">
186   <DA name="vendor" bType="VisString255" fc="DC" />
187 </DOType>
188 - <DOType id="ASG_2_StrVal" cdc="ASG">
189   <DA name="minVal" type="AnalogueValue_2" bType="Struct" fc="CF" />
190 </DOType>
191 - <DOType id="LPL_2_NamPlt" cdc="LPL">
192   <DA name="vendor" bType="VisString255" fc="DC" />
193   <DA name="swRev" bType="VisString255" fc="DC" />
194   <DA name="d" bType="VisString255" fc="DC" />
195 </DOType>
196 - <DOType id="SPC_1_BlkOpn" cdc="SPC">
197   <DA name="Cancel" type="SPCCancel_1" bType="Struct" fc="CO" />
198   <DA name="stVal" bType="BOOLEAN" fc="ST" dchg="true" />
199   <DA name="q" bType="Quality" fc="ST" qchg="true" />
200   <DA name="t" bType="Timestamp" fc="ST" />
201   <DA name="subEna" bType="BOOLEAN" fc="SV" />
202   <DA name="subVal" bType="BOOLEAN" fc="SV" />
203   <DA name="subQ" bType="Quality" fc="SV" />
204   <DA name="subID" bType="VisString64" fc="SV" />
205   <DA name="ctlModel" type="ctlModelEnum" bType="Enum" fc="CF" />
206 </DOType>
207 - <DOType id="DPC_1_Pos" cdc="DPC">
208   <DA name="Cancel" type="DPCCancel_1" bType="Struct" fc="CO" />
209   <DA name="stVal" bType="Dbpos" fc="ST" dchg="true" />
210   <DA name="q" bType="Quality" fc="ST" qchg="true" />
211   <DA name="t" bType="Timestamp" fc="ST" />
212   <DA name="subEna" bType="BOOLEAN" fc="SV" />
213   <DA name="subVal" bType="Dbpos" fc="SV" />
214   <DA name="subQ" bType="Quality" fc="SV" />
215   <DA name="subID" bType="VisString64" fc="SV" />
216   <DA name="ctlModel" type="ctlModelEnum" bType="Enum" fc="CF" />
217 </DOType>
218 - <DOType id="MV_1_Vol" cdc="MV">
219   <DA name="instMag" type="AnalogueValue_1" bType="Struct" fc="MX" />
220   <DA name="mag" type="AnalogueValue_1" bType="Struct" fc="MX" dchg="true" />
221   <DA name="q" bType="Quality" fc="MX" qchg="true" />
222   <DA name="t" bType="Timestamp" fc="MX" />
223   <DA name="subEna" bType="BOOLEAN" fc="SV" />
224   <DA name="subMag" type="AnalogueValue_1" bType="Struct" fc="SV" />
225   <DA name="subQ" bType="Quality" fc="SV" />
226   <DA name="subID" bType="VisString64" fc="SV" />
227   <DA name="db" bType="INT32U" fc="CF" />
228 </DOType>
229 - <DOType id="LPL_3_NamPlt" cdc="LPL">
230   <DA name="vendor" bType="VisString255" fc="DC" />

```

```

231 <DA name="swRev" bType="VisString255" fc="DC" />
232 <DA name="d" bType="VisString255" fc="DC" />
233 </DOType>
234 - <DOType id="INS_2_Beh" cdc="INS">
235 <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
236 <DA name="q" bType="Quality" fc="ST" qchg="true" />
237 <DA name="t" bType="Timestamp" fc="ST" />
238 <DA name="subEna" bType="BOOLEAN" fc="SV" />
239 <DA name="subVal" bType="INT8" fc="SV" />
240 <DA name="subQ" bType="Quality" fc="SV" />
241 <DA name="subID" bType="VisString64" fc="SV" />
242 </DOType>
243 - <DOType id="INC_2_Mod" cdc="INC">
244 <DA name="Cancel" type="INCCancel_2" bType="Struct" fc="CO" />
245 <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
246 <DA name="q" bType="Quality" fc="ST" qchg="true" />
247 <DA name="t" bType="Timestamp" fc="ST" />
248 <DA name="subEna" bType="BOOLEAN" fc="SV" />
249 <DA name="subVal" bType="INT8" fc="SV" />
250 <DA name="subQ" bType="Quality" fc="SV" />
251 <DA name="subID" bType="VisString64" fc="SV" />
252 <DA name="ctlModel" type="ctlModelEnum" bType="Enum" fc="CF" />
253 </DOType>
254 - <DOType id="ASG_1_StrVal" cdc="ASG">
255 <DA name="minVal" type="AnalogueValue_2" bType="Struct" fc="CF" />
256 <DA name="maxVal" type="AnalogueValue_2" bType="Struct" fc="CF" />
257 </DOType>
258 - <DOType id="ACT_2_Op" cdc="ACT">
259 <DA name="general" bType="BOOLEAN" fc="ST" dchg="true" />
260 <DA name="q" bType="Quality" fc="ST" qchg="true" />
261 <DA name="t" bType="Timestamp" fc="ST" />
262 </DOType>
263 - <DOType id="ACD_2_Str" cdc="ACD">
264 <DA name="general" bType="BOOLEAN" fc="ST" dchg="true" />
265 <DA name="dirGeneral" type="dirGeneralEnum" bType="Enum" fc="ST" dchg="true" />
266 <DA name="q" bType="Quality" fc="ST" qchg="true" />
267 <DA name="t" bType="Timestamp" fc="ST" />
268 </DOType>
269 - <DOType id="LPL_4_NamPlt" cdc="LPL">
270 <DA name="vender" bType="VisString255" fc="DC" />
271 <DA name="swRev" bType="VisString255" fc="DC" />
272 <DA name="d" bType="VisString255" fc="DC" />
273 </DOType>
274 - <DOType id="INS_3_Beh" cdc="INS">
275 <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
276 <DA name="q" bType="Quality" fc="ST" qchg="true" />
277 <DA name="t" bType="Timestamp" fc="ST" />
278 <DA name="subEna" bType="BOOLEAN" fc="SV" />
279 <DA name="subVal" bType="INT8" fc="SV" />
280 <DA name="subQ" bType="Quality" fc="SV" />
281 <DA name="subID" bType="VisString64" fc="SV" />
282 </DOType>
283 - <DOType id="INC_3_Mod" cdc="INC">
284 <DA name="Cancel" type="INCCancel_3" bType="Struct" fc="CO" />
285 <DA name="stVal" bType="INT8" fc="ST" dchg="true" />
286 <DA name="q" bType="Quality" fc="ST" qchg="true" />
287 <DA name="t" bType="Timestamp" fc="ST" />
288 <DA name="subEna" bType="BOOLEAN" fc="SV" />
289 <DA name="subVal" bType="INT8" fc="SV" />
290 <DA name="subQ" bType="Quality" fc="SV" />
291 <DA name="subID" bType="VisString64" fc="SV" />
292 <DA name="ctlModel" type="ctlModelEnum" bType="Enum" fc="CF" />

```

```

293 </DOType>
294 - <DAType id="INCCancel_1">
295   <BDA name="ctlVal" bType="INT8" />
296   <BDA name="origin" type="Originator_1" bType="Struct" />
297   <BDA name="ctlNum" bType="INT8U" />
298   <BDA name="T" bType="Timestamp" />
299   <BDA name="Test" bType="BOOLEAN" />
300 </DAType>
301 - <DAType id="Originator_1">
302   <BDA name="orCat" type="orCatEnum" bType="Enum" />
303   <BDA name="orIdent" bType="Octet64" />
304 </DAType>
305 - <DAType id="SPCCancel_1">
306   <BDA name="ctlVal" bType="BOOLEAN" />
307   <BDA name="origin" type="Originator_1" bType="Struct" />
308   <BDA name="ctlNum" bType="INT8U" />
309   <BDA name="t" bType="Timestamp" />
310   <BDA name="Test" bType="BOOLEAN" />
311 </DAType>
312 - <DAType id="DPCCancel_1">
313   <BDA name="ctlVal" bType="BOOLEAN" />
314   <BDA name="origin" type="Originator_1" bType="Struct" />
315   <BDA name="ctlNum" bType="INT8U" />
316   <BDA name="t" bType="Timestamp" />
317   <BDA name="Test" bType="BOOLEAN" />
318 </DAType>
319 - <DAType id="AnalogueValue_1">
320   <BDA name="f" bType="FLOAT32" />
321 </DAType>
322 - <DAType id="INCCancel_2">
323   <BDA name="ctlVal" bType="INT8" />
324   <BDA name="origin" type="Originator_1" bType="Struct" />
325   <BDA name="ctlNum" bType="INT8U" />
326   <BDA name="t" bType="Timestamp" />
327   <BDA name="Test" bType="BOOLEAN" />
328 </DAType>
329 - <DAType id="Originator_3">
330   <BDA name="orCat" type="orCatEnum" bType="Enum" />
331   <BDA name="orIdent" bType="Octet64" />
332 </DAType>
333 - <DAType id="Originator_2">
334   <BDA name="orCat" type="orCatEnum" bType="Enum" />
335   <BDA name="orIdent" bType="Octet64" />
336 </DAType>
337 - <DAType id="AnalogueValue_2">
338   <BDA name="f" bType="FLOAT32" />
339 </DAType>
340 - <DAType id="INCCancel_3">
341   <BDA name="ctlVal" bType="INT8" />
342   <BDA name="origin" type="Originator_1" bType="Struct" />
343   <BDA name="ctlNum" bType="INT8U" />
344   <BDA name="t" bType="Timestamp" />
345   <BDA name="Test" bType="BOOLEAN" />
346 </DAType>
347 - <DAType id="Originator_4">
348   <BDA name="orCat" type="orCatEnum" bType="Enum" />
349   <BDA name="orIdent" bType="Octet64" />
350 </DAType>
351 - <EnumType id="orCatEnum">
352   <EnumVal ord="0">not-supported</EnumVal>
353   <EnumVal ord="1">bay-control</EnumVal>
354   <EnumVal ord="2">station-control</EnumVal>

```

```

355 <EnumVal ord="3">remote-control</EnumVal>
356 <EnumVal ord="4">automatic-bay</EnumVal>
357 <EnumVal ord="5">automatic-station</EnumVal>
358 <EnumVal ord="6">automatic-remote</EnumVal>
359 <EnumVal ord="7">maintenance</EnumVal>
360 <EnumVal ord="8">process</EnumVal>
361 </EnumType>
362 - <EnumType id="dirGeneralEnum">
363 <EnumVal ord="0">unknown</EnumVal>
364 <EnumVal ord="1">forward</EnumVal>
365 <EnumVal ord="2">backward</EnumVal>
366 <EnumVal ord="3">both</EnumVal>
367 </EnumType>
368 - <EnumType id="ctlModelEnum">
369 <EnumVal ord="0">status-only</EnumVal>
370 <EnumVal ord="1">direct-with-normal-security</EnumVal>
371 <EnumVal ord="2">sbo-with-normal-security</EnumVal>
372 <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
373 <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
374 </EnumType>
375 - <EnumType id="rangeEnum">
376 <EnumVal ord="0">normal</EnumVal>
377 <EnumVal ord="1">high</EnumVal>
378 <EnumVal ord="2">low</EnumVal>
379 <EnumVal ord="3">high-high</EnumVal>
380 <EnumVal ord="4">low-low</EnumVal>
381 </EnumType>
382 </DataTypeTemplates>
383 </SCL>

```