Technical University of Denmark

A Thesis Submitted for the Degree

Master of Science

Reporting and Logging in compliance with IEC 61400-25

Using Wind Power Plant Configuration Language (WPPCL)

Author:
s011903, Umut Korkmaz

Supervisors:
Bjarne Poulsen
Knud Johansen

July 2007
Kongens Lyngby, Denmark

Abstract

Vendors and customers in the wind power plant market are not capable of choosing each other purely based on business related parameters. Rather, they are constrained by different approaches for modeling wind power plants in the software domain. IEC 61400-25 addresses this challenge. This thesis analyzes, designs and implements a proof of concept system capable of reporting and logging in compliance with IEC 61400-25. The system consists of an information model, an information exchange model and is mapped to web services. Both unbuffered and buffered reporting is supported. Reporting uses publisher/subscriber by use of the WSDualHttpBinding provided in Windows Communication Foundation (WCF). Logging is realized by use of persistent storage. A client with SCADA capable of consuming the services exposed by the system has been created. Exchange of information between system and client follows the guidelines for SOA. Contents of the data model is configured by use of xml based Wind Power Plant Configuration Language (WPPCL). A tool named WPPCL Editor for configuring the WPPCL file has been created. Wind power plants are modeled by a simple data generator in order to verify the functionality of the system in its natural environment.

Acknowledgements

First, I thank my supervisor Bjarne Poulsen, for his continuous support. Second, I thank Knud Johansen for his insightful feedback regarding IEC 61400-25 and wind power plants in general. I also thank Knud Ole Helge Pedersen for listening and providing feedback.

Contents

List of Figures

List of Tables

Abbreviations

Control blocks: UBRCB, BRCB and LCB
SCADA: Supervisory Control And Data Acquisition
SOA: Service Oriented Architecture
Ssd: System Sequence Diagram
Ucr: Use Case Realization
WCF: Windows Communication Foundation
WPPCL: Wind Power Plant Configuration Language

Chapter 1

Introduction

Creating a proof of concept system in compliance with the IEC 61400-25 standard is the main objective of this thesis. Attention will be on the reporting and logging parts of the standard. Both these subjects are in the monitoring category of the standard. In order to use the system, a client with SCADA will be created. Wind Power Plant Configuration Language (WPPCL) will be used to create an xml file (WPPCL file) that specifies the contents of the data model in the system. The WPPCL file will be used to initialize the contents of the data model in the system. An editor named WPPCL Editor will be created in order to edit the WPPCL file, thus configuring the contents of the data model in the system.

This chapter presents the background and motivation for the thesis in sections 1.1 and 1.2, respectively. Section 1.3 presents the vision for the thesis. Section 1.4 presents the project description. The chapter ends with a presentation of the report outline in section 1.5.

1.1 Background

Readiness for change is a key factor for companies in order to handle the changing internal and external challenges in business. Companies in the wind power plant market are no exception to this.

Wind power plants need to be modeled in the software domain in order to be monitored and controlled by external actors. An external actor can be the owner of the wind power plant or a customer who bought energy from the wind power plant. Modeling of the wind power plant has a vendor side (server) and a customer side (client).

The key question is how to model the wind power plant in the software domain. Every vendor can make its own server solution and the customers that cooperate with this vendor can create their own client solutions. The drawback of this approach is that it creates tight coupling between vendor and customer. This approach is the scenario in the wind power plant market today. It limits the degree of readiness for change.

IEC 61400-25 addresses this challenge. The purpose of IEC 61400-25 is to ”provide a uniform communication basis for the monitoring and control of wind power plants” [61400-25-1]. Reporting and logging are included in the

monitoring part.

The standard presents a client and server, defines what they shall communicate and how they shall communicate. How the server communicates with the wind power plant is outside scope in the standard.

IEC 61400-25 consists of five parts. First part [61400-25-1] is an introduction describing the overall scenario. Second part [61400-25-2] defines the information model of wind power plants. This includes how a wind power plant must be modeled in the software domain. Part three [61400-25-3] describes which services must be available for the client and server in order to exchange the information. Part four [61400-25-4] presents different mappings to protocol stacks. Web services being one of the options for mapping will be used in this thesis. Part five [61400-25-5] is last part and defines testing. Although testing will be done in this thesis, [61400-25-5] will not be used to do it.

The standard consists of mandatory elements and optional elements. Reporting and logging are among the optional elements. Although the data model of a IEC 61400-25 has a defined structure, its contents can differ due to the optional elements in the standard. Thus, only the structure for the data model must be a part of an IEC 61400-25 compliant system, rather than the contents of the data model. This can be achieved by use of WPPCL, which specifies the contents of an IEC 61400-25 compatible wind power plant. The hierarchical structure of the data model in IEC 61400-25 can be reflected by use of xml. The WPPCL file is used by the system to set the contents of its data model in order to reflect the data model of the modeled wind power plant.

Not every element of the data model of a given wind power plant may be relevant for a given system. This is why it must be possible to edit the WPPCL file. The purpose of WPPCL Editor is to edit the WPPCL file.

In order to have the system working in a natural environment, ideally a wind power plant must be used. This has not been an option in this thesis, which is why a wind power plant data generator (WPP data generator) is used.

Former work in the area has been done in [Andreas & Baris] with purpose to evaluate the major parts of IEC 61400-25 and implement a working system. It must be noted that while the work in [Andreas & Baris] was being done, the IEC 61400-25 was still under progress. At this moment part one, two, three and five of the standard are stabilized. Part four is still under progress.

Besides [Andreas & Baris] there is not a lot of work done in the area. This can be seen due to the fact that IEC 61400-25 is a relative new standard.

1.2 Motivation

The motivation for this thesis is to implement an IEC 61400-25 compliant proof of concept system with attention on the monitoring part in order to show how the specifications of IEC 61400-25 can result in a operational monitoring system. The motivation for using the web services mapping is considered due to the wide use of the Internet.

The motivation for using a WPPCL file is to ensure that the system will be able to model all IEC 61400-25 compatible wind power plants, rather than being tied to a specific wind power plant. IEC 61400-25 has a lot of optional elements, and consequently an IEC 61400-25 compliant system must be able to handle with all these variations.

The need for WPPCL Editor can be seen in contrast to the alternative. The alternative is to edit the WPPCL file in a text editor, since it is an xml file. However this is a potential error prone task. Leaving the WPPCL file in non valid state after an edit, will propagate to the system, which will fail reading the WPPCL file, thus failing to function correctly.

The WPP data generator is necessary, because the system will eventually be used with real wind power plants. By use of the data generator, the functionality of the system can be verified and tested as if real wind power plants were used.

1.3 Vision

The vision for this thesis is to free the actors in the wind power plant market from using proprietary solutions for modeling wind power plants, thus achieving higher degree of freedom for choosing who to work with. By use of IEC 61400-25 they will be able to cooperate with each other based on business related parameters, rather than letting proprietary modeling of wind power plants be the limiting major parameter. In the long run the level of readiness for change will be increased. This will have a positive effect internally in the wind power plant market, thus making wind energy more competitive against outside competitors such as the oil industry.

1.4 Project description

This thesis will analyze, design, implement and test reporting and logging as specified in IEC 61400-25. Both unbuffered and buffered reporting is considered. The outcome of the project will be

- An IEC 61400-25 compliant system with focus on reporting and logging that exposes its information in terms of web services
- A client including SCADA that is able to consume the services exposed by the system
- A WPPCL file defining the contents, in terms of data, of an imaginary wind power plant
- WPPCL Editor which makes it possible to edit the WPPCL file
- A data generator, which seen from the perspective of the system, is a wind power plant that generates data.

Focus is on the monitoring part of IEC 61400-25 and details not relevant in this regard will be left out. For instance, controlling the wind power plant is not within focus of the thesis.

The system makes its services available in terms of web services, in order for the services to be consumed by use of the Internet. The publisher/subscriber pattern is used for reporting. With unbuffered reporting, if the client has not established a connection to the system, the reports are discarded. With buffered reporting, the reports are buffered.

The WPPCL file is used by the system at initialization to determine the contents of its data model. The WPPCL file is edited by use of the WPPCL Editor, which removes the risk of errors due to manual editing.

After initialization, the system polls the data generator in order to retrieve data from a wind power plant. The data generator generates random data, rather than generating realistic wind power plant data. A realistic WPP data generator can be created in collaboration with people who have insight and knowledge about wind power plants, which is not true for the writer of this thesis. An actual wind power plant could also replace the data generator. This topic is left open for future work.

1.5 Report outline

Major parts of the report are organized into chapters. Chapter 2 analyzes relevant parts for the thesis. This includes how IEC 61400-25 specifies reporting and logging. Subjects out of scope for the standard such as the WPPCL file and the WPPCL Editor will also be analyzed. The analysis results in a requirements specification defining the foundation for the rest of the thesis.

Chapter 3 designs a system and a client with SCADA that meets the requirements defined in the analysis. It also designs the WPPCL file and WPPCL Editor.

Chapter 4 constructs the system, client with SCADA, WPPCL file and WPPCL Editor.

Chapter 5 tests the IEC 61400-25 compliant system by use of the client. The WPPCL Editor is also tested.

Chapter 6 concludes the thesis with a presentation and discussion of the results besides suggestions for future work.

Chapter 2

Analysis

The purpose of this chapter is to analyze the components of the thesis. A system must be created that provides reporting and logging in compliance with IEC 61400-25. This system is analyzed in section 2.1.

Out of scope subjects for IEC 61400-25 but important in order to build a complete monitoring system consists of following subjects

- WPPCL file
- WPPCL Editor
- WPP data generator

Wind power plants from different vendors vary by the contents of their data model. In this thesis, only IEC 61400-25 compatible wind power plants in terms of structure for data model are considered. This means that wind power plants that the system has to model, must have a data model with structure as defined in [61400-25-2] with the server element in the top down to the data attribute element in the bottom. The modeled wind power plants can vary in the contents of their data model, not in the structure of the data model. In order to reflect the contents of a given data model in the system, a language for describing such data model is expressed in a standardized fashion with WPPCL in a WPPCL file. The file is used to initialize the contents of the data model in the system. Thus, the system will be able to model all IEC 61400-25 compatible wind power plants. The structure and format for the WPPCL file is analyzed in section 2.2. The WPP data generator will be created as part of the system because no real wind power plant is used. The WPP data generator will be used by the system to poll for data at regular intervals.

It must be possible to customize the contents of the data model in the system because not every system might be interested in the entire contents of the data model that a given wind power plant provides. This is achieved by editing the WPPCL file. A WPPCL editor is needed in order to support intuitive editing of the WPPCL file. WPPCL Editor is analyzed in section 2.3.

A client that is capable of using the services exposed by the system must be created in order to demonstrate the behavior of the system. To provide interaction with humans the client must have a graphical user interface, representing a

simple SCADA. By interaction with the SCADA it must be possible to configure and use the reporting and logging, besides general use of the system. Client including SCADA is the subject of section 2.4.

The chapter results in a formal requirements specification in section 2.5 defining the requirements for the thesis. Conclusion for the analysis is last part and can be found in section 2.6.

2.1 IEC 61400-25 compliant system

The IEC 61400-25 compliant system consists of

- Information model
- Information exchange model
- Mapping to web services

The information model of the system is specified in [61400-25-2] and it defines the information that must be possible for client and system to exchange. This includes the report control blocks (unbuffered and buffered) and the log control block. It also includes data sets and the data model of the system, which consists of a hierarchical structure from the server element in top to the data attribute element in the bottom. The information model is analyzed in section 2.1.1.

The information exchange model specified in [61400-25-3] defines the methods that the client and system must use in order to exchange information (hence the name) contained in the information model. Section 2.1.2 analyzes the information exchange model.

The methods defined in the information exchange model are abstract and can be mapped (implemented) to different protocol stacks. The different mappings are presented in [61400-25-4]. This thesis will use the web service mapping. Section 2.1.3 analyzes the concept of mapping. It also describes the chosen mapping environment for the thesis, namely Windows Communication Foundation (WCF) and how this can be used to follow the guidelines for Service Oriented Architecture (SOA).

The system must poll data from a data generator in order to simulate a natural environment. The data generator is the subject of section 2.1.4.

The process of determining if reporting and logging must occur is similar. Describing the process for determining if reporting and logging must happen is the subject of section 2.1.5. Sections 2.1.6 and 2.1.7 describe the reporting and logging mechanisms, respectively.

Section 2.1.8 identifies and visualizes objects with their relationships in the domain model for the system. This will serve as inspiration when designing the system.

Requirements to the system, seen from the perspective of the client, is expressed as use cases in section 2.1.9. The use case diagram in section 2.1.10 presents a visual overview for the identified use cases. In section 2.1.11 system sequence diagrams for the identified use cases are presented in order to visualize interaction between client and system.

2.1.1 Information model

[61400-25-2] presents the information model which defines a hierarchical structure for the data model. The data model is modeling components from a real wind power plant in the software domain. The term functional constraint is used to address specific data attributes and the term trigger option is used in the process for determining if reporting and logging must occur.

In addition [61400-25-2] defines the structure for data sets, log control block (LCB), unbuffered report control block (UBRCB) and buffered report control block (BRCB).

Data model

Explanation of the data model within this thesis is only relevant seen from the perspective of building an object oriented system. In other words, only subjects relevant for the software domain will be explained. Thus it is not of interest, how the data model corresponds to components in real wind power plants or details related to the wind power plant domain in general.

The system has an internal representation of the information model as a hierarchy. At the very top the server element has its place. The server is unique, that is, only one server element per IEC 61400-25 compliant system. The server can hold one or multiple logical devices. The logical device represents a wind power plant. Each logical device can hold one or multiple logical nodes. The logical nodes represent components of the wind power plant, such as the turbine (WTUR). Each logical node can hold one or multiple data entities. Each data entity can hold one or multiple data groups. Each data group can hold one or multiple data attributes. Data attributes are the smallest building block in the data model. They consist of simple types such as integers or Booleans. The hierarchy of the data model can be seen in figure 2.1.

As an example, the specification for logical node WTUR (Wind turbine general information) is considered from [61400-25-2]. The first data entity for WTUR is named AvlTmsRs (turbine availability time). AvlTmsRs is of type TMS (state timing). TMS is a Common Data Class (CDC). CDC's group together common data, which can be used by various logical nodes. The specification for TMS is also localized in [61400-25-2]. The first entry for TMS is data attribute ctlVal. As mentioned before, the data attribute is the most basic building block. However ctlVal is present more than once in TMS. It is present under "manRs" and "hisRs". An additional abstraction level between the data entity and data attribute is needed in order to address data attributes with a unique path. This abstraction level is named data group in this thesis. Assuming for this example that the data is in a logical device named LD1, the data attribute can now be referenced with following unique path

```
LD1.WTUR.AvlTmsRs.manRs.ctVal
```

or more generally

```
LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute
```

It is possible to make use of a concept such as the tree to visualize the information model due to the hierarchical structure. However it is not the responsibility of the system to visualize the data model. This is up to the client.

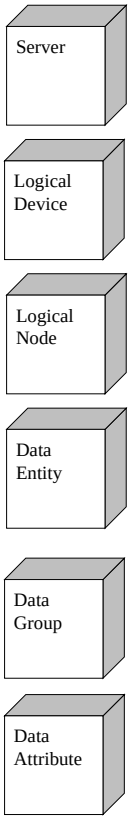

Figure 2.1: Data model is hierarchical

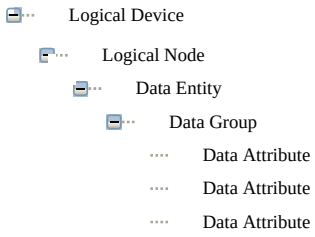

Figure 2.2: Visualization of general structure for data model

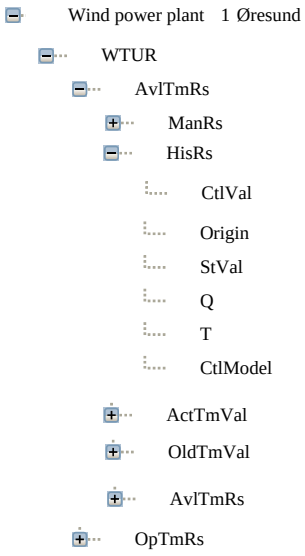

Figure 2.3: Concrete example for data model structure

The general structure for such a tree can be seen in figure 2.2 and a concrete example of the tree can be seen in figure 2.3.

The contents of the data model in the system is a reflection of the contents of the data model of the modeled wind power plant, represented by the WPPCL file. It is not possible for the data model in the system to contain more data than specified by the WPPCL file. Turning it the other way around, it is possible for the system to contain less data than specified initially in the WPPCL file. This is achieved by editing the WPPCL file, by use of WPPCL Editor.

Data attributes are the lowest level of information in IEC 61400-25. They are characterized by attribute name, attribute type, functional constraint, trigger option, explanation/range and mandate as can be seen in table 2.1. To view the complete collection of data attributes, refer to [61400-25-2]. Note that IEC 61400-25 also inherits data attributes from [61850-7-2]. The structure for the data attributes, however, is the same. Attribute name and attribute type defines the name of the data attribute and the type of information that it holds. For instance the data attribute ”t” has type ”timestamp”. Functional constraint and trigger option will be described below. Explanation/range describes the data

attribute together with its range. Mandate specifies whether the given data attribute is mandatory or optional.

Functional Constraint

The functional constraint specifies which operations are allowed on the data attribute. If data attribute stVal is considered it can be seen in [61400-25-2] that it has the functional constraint ST (status value). This functional constraint specifies that the data attribute must be possible to be read, substituted, reported and logged. Writing to the data attribute is not allowed. An overview and explanation of the different functional constraints can be found in table 18 in [61850-7-2].

However, the main use of the functional constraint in this thesis is related to creation of data sets as will be described later in this section.

Trigger Option

The trigger option specifies whether the data attribute is capable of triggering reporting and logging. Three types of triggers exist. These are

- dchg (data change)
- qchg (quality change)
- dupd (data update)

As an example, data attribute stVal is associated with the trigger dhcg. This means that every time data changes for this data attribute reporting or logging can potentially occur. Whether it happens or not depends on the state of subscriptions for the data attribute. The process for determining if reporting and logging must happen is described in section 2.1.5.

Data Set

Reporting related logical node WREP and logging related logical nodes WSLG and WALG operates with an abstraction level named data set that groups together references for data attributes. Reporting and logging happens at data set level rather than data attribute level.

Note that only references to the data attributes are grouped together rather than the actual data attributes. Figure 2.4 illustrates the concept of a data set grouping together references for data attributes.

Property	*Description*
Attribute name	Name of attribute
Attribute type	Int, Boolean, string etc.
Functional constraint	Which operations are supported
Trigger option	Used for reporting and logging
Explanation/range	Description and range of attribute content
Mandate	Mandatory or optional?

Table 2.1: Data attribute properties

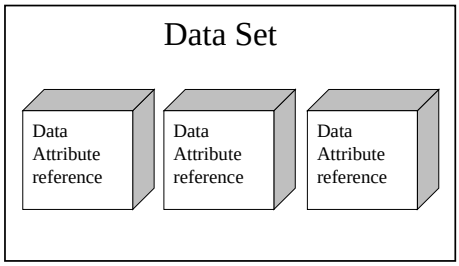

Figure 2.4: Data set groups together references for data attributes

Initially data sets have no references for data attributes. They must be created. Two ways of creating data sets exist. Either they are preconfigured (and created at system startup) or they can be dynamically created in the lifetime of the system. With the first approach, the client has no influence on the data sets. With dynamically created data sets, the client can configure the data sets in the lifetime of the system.

For both preconfigured and dynamically created data sets, references for data attributes depend on the contents of the data model in the system. Consequently data sets can be created only after the contents of the data model in the system has been initialized.

The process of creating data sets is similar for preconfigured and dynamic data sets. The process consists of scanning the data model for the data attributes as specified by the rules of the data set. Every time a match is found, the data attribute reference is added to the data set. The rules for a data set can have two formats

- CDC and data attribute
- CDC, data group and functional constraint

References for data attributes use the unique path, as mentioned before, that is

`LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute`

The key difference between preconfigured and dynamic data sets is when the data sets are created. With preconfigured data sets, the data sets are created only at system startup according to the rules specified in [61400-25-2]. With dynamic data sets, rules for data sets can change in the lifetime of the system. Thus the system must be capable of updating the data set contents when rules change by searching the data model and updating references. Because rules for preconfigured data sets is possible to change, the client is able to configure data set dynamically.

However since the process for creating data sets are equivalent for preconfigured and dynamic data sets (search the data model and add references when a match is found), for simplicity's sake only preconfigured data sets will be used in this thesis. The definitions for the preconfigured data sets can be found in [61400-25-2], and will be presented below.

Data sets for reporting (WREP) consist of

- TurRpCh: Data attributes ”mag” that are to be found in CDC MV.
- TurRpTm: Data attributes ”dly”, ”mly”, ”yly” and ”tot” in CDC TMS.
- TurRpCt: Data attributes ”dly”, ”mly”, ”yly” and ”tot” in CDC CTE.

The data attribute references defined in WREP as can be seen above uses the rules defined in the format (CDC, data attribute). Considering TurRpCb, its rules says that every data attribute that has the name ”mag” which is located in the CDC named MV must be referenced by the data set named TurRpCb. The explanation is similar for TurRpTm and TurRpCt.

Data sets for logging consist of data sets from the two logical nodes WSLG and WALG. In order to view the data set names and their rules, refer to appendix A.1 for WSLG and appendix A.2 for WALG. However, the data set named TurCtLog will be explained here, because it uses the second format for expressing rules, that is, by (CDC, data group, functional constraint). The rule for TurCtLog is (CTE, actCtVal, ST). This means that every data attribute that is within the datagroup named actCtVal in the CDC named CTE *and* has the functional constraint ST must be referenced by the data set named TurCtLog.

Unbuffered Report Control Block (UBRCB)

The client uses UBRCB to express its interest in unbuffered reporting by subscribing to reporting related data sets.

UBRCB is responsible for reporting to the client, when data attributes in the subscribed data sets satisfy the conditions for reporting. The conditions for reporting (and logging) is described in section 2.1.5.

An alternative to subscription at data sets level would be to have them at data attribute level. However this would be too low level control introducing too much subscription/unsubscription work for the client.

UBRCB is per client basis. That is, each client has its own UBRCB. This ensures that each client can configure its own unbuffered reporting.

The complete specification for UBRCB can be seen in table 25 in [61850-7-2]. However in order to keep it simple only the following attributes of the UBRCB will be used in this thesis

- UBRCBName
- RtpEna
- DatSet
- Report-time-stamp

UBRCBName is the unique name for UBRCB which is used to identify which client it belongs to. The RtpEna indicates if the UBRCB is enabled or disabled. It must be enabled in order for unbuffered reporting to happen. DatSet holds the references for the subscribed data sets. Report-time-stamp defines when the report was generated.

Buffered Report Control Block (BRCB)

The BRCB is used when buffered reporting is intended. The complete specification for BRCB can be found in table 23 in [61850-7-2]. As with UBRCB, this thesis will not use the complete specification for BRCB. Following attributes of the BRCB will be used

- BRCBName
- RtpEna
- DatSet
- Report-time-stamp
- Report id

Descriptions for the attributes are similar to the UBRCB. However the report id is only used with BRCB. It is a unique id that every report must have. Every report gets its own unique id in chronological order according to the time they are generated. The purpose of the report id is to make it possible for the client to know, if it has received all reports or if some is missing. The client will also be able to verify that reports are delivered in chronological order. BRCB must have a buffer for buffering the reports.

Log Control Block (LCB)

LCB is used for logging. It has the responsibility to log data attributes whenever conditions for logging have been satisfied.

A complete specification of LCB can be found in table 26 in [61850-7-2], but only following attributes will be used in the thesis.

- LCBName
- LogEna
- DatSet

LCBName is a unique name used to identify which client a given LCB belongs to. Like UBRCB and BRCB, LCB is per client basis. This ensures that every client is capable of customizing its own logging. LogEna indicates if the LCB is enabled or disabled. Only if it is enabled logging can occur. The attribute DatSet defines which data sets must be logged.

2.1.2 Information exchange model

[61400-25-3] presents the information exchange model. The model defines which services the client and system is able to invoke in order to exchange the data contained in the information model. The services include reporting and logging. The model is abstract (Abstract Communication Service Interface (ACSI) and does not put implementation specific constraints on the services.

Relevant services that the client and system have to use in order to access the information model can be seen in table B.1 in [61400-25-3]. Not every service from the standard will be provided by the information exchange model in this

thesis. For instance the service SetDataSetValues is not exposed, because the client is not able to configure the data sets. Data sets are created internally within the system, based on predefined rules. However the system uses the service SetDataSetValues internally while creating the data sets, and it would be straightforward to expose the service in the information exchange model. Then the client would be able to configure the data sets. In such scenario, however, it must be considered, if data sets must be per client rather than per system, in order not to change data set contents for other clients.

The services of the information exchange model can be grouped according to their purpose. As will become apparent in section 2.1.9, the groupings will inspire while identifying and creating use cases. Figure 2.5 provides an overview for the services and their groupings. The services of the information exchange model with groupings will be described below.

Association

Association is used to identify a client to the system. This ensures that the client gets its own UBRCB, BRCB and LCB. The client has a unique id, which is used for the association. In a real world scenario authentication of the clients and a central policy for issuing id to the clients would be necessary. A simple approach will be used in this thesis, where each client gets its own unique id. Association will take place by use of the client id only. No secure authentication mechanism, such as typing in password or using an encrypted key file will be used. This can be a subject for access control in future works.

Retrieve data model contents

In order to retrieve contents of the data model in the system, following services, grouped as RetrieveDataModelContents, can be used by the client

- GetServerDirectory
- GetLogicalDeviceDirectory
- GetLogicalNodeDirectory
- GetDataEntityDirectory
- GetDataDirectory
- GetDataValues

The service GetServerDirectory returns all the logical devices contained in the system. GetLogicalDeviceDirectory returns all the logical nodes contained in a particular logical device, specified by the client. GetLogicalNodeDirectory returns all the data entities contained in a particular logical node. The service GetDataEntityDirectory is used to return all data groups in a given data entity. The service GetDataDirectory is used to retrieve the data attributes within a data group. Note that the GetDataValues will not be used in this thesis because it is not directly involved with reporting and logging. The reason that it has been presented, is because it is closely related with the other services for retrieving contents of the data model.

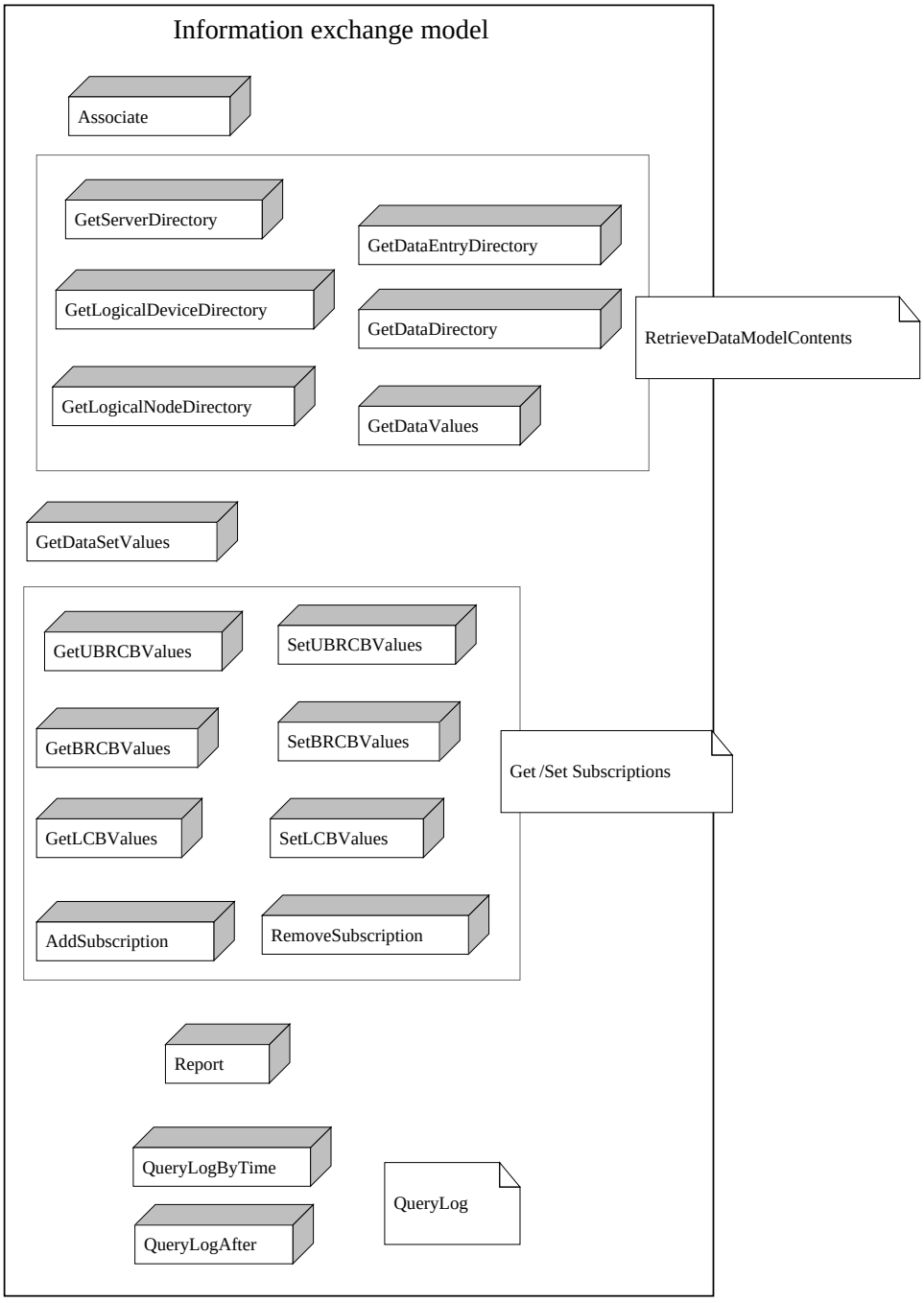

Figure 2.5: Information exchange model with grouped services

Data Set

The service GetDataSetValues returns the data attributes that are referenced by a given data set. Note that the service SetDataValues is not a part of the information exchange model. This is due to the fact that preconfigured data sets are used, thus the client will not be able to configure data sets. However, as will be presented later, the system internally uses the SetDataSetValues method when creating data sets by use of predefined rules. If the client must be able to configure data sets, exposing this method as a service would do the job.

Get/Set Subscriptions

The Get/Set Subscriptions applies to the control blocks (UBRCB, BRCB and LCB), where subscriptions for data sets exist. Although according to [61400-25-3] the services AddSubscription and RemoveSubscription only applies for the reporting mechanism, they will be used for logging as well. Regarding these two services, they are considered as high level services. This means that they consist of two low level operations. First step is to have a data set (either create it or use an existing) and second step consists of invoking the Set[ControlBlock]Values service and give it the particular data set. With AddSubscription reference to the data set is added. With RemoveSubscription the existing data set reference is removed. The service Set[ControlBlock]Values is used both for adding subscriptions and removing subscriptions. The parameters given to the service determine whether to add or remove a subscription.

The reason that AddSubscription and RemoveSubscription applies to both reporting and logging is due to the mechanism of expressing interest in reporting and logging. For both mechanisms it happens at data set level. The client says that it wants reporting/logging to happen for a particular data set by subscribing to it. Only difference between reporting and logging is the outcome of the subscriptions. While reporting reports (or buffers), the logging mechanism logs data entries.

The services for retrieving a list of current subscriptions is obtained by the Get[ControlBlock]Values. For instance the service GetUBRCBValues will return the list of subscribed data sets for the UBRCB.

To summarize, the services used for getting and setting subscriptions are

- GetUBRCBValues
- SetUBRCBValues
- GetBRCBValues
- SetBRCBValues
- AddSubscription
- RemoveSubscription

It must be noted that [61400-25-3] does not limit the Get and Set services operating on Control Blocks to only handle subscriptions. Other operations such as enabling or disabling a Control Block are also handled with the Set service for the particular control block. However, since reporting and logging are the main topics in this thesis, the subscription parameter (in terms of data sets) of the

control blocks has been considered as the primary object for using the Get/Set operations on the control blocks. This is the reason why the services are grouped together as Get/Set Subscriptions rather than Get/Set [ControlBlock]Values. Additional services will be used for enabling and disabling the control blocks.

The services AddSubscription and RemoveSubscription will not be used explicitly in the thesis, because behind the scenes, they use the services Set[ControlBlock]Values.

Report

The report service is used by the system uses in order to deliver reports spontaneously to the client. All the client has to do in order to retrieve reports is to subscribe to relevant data sets. However, reporting is not guaranteed, because conditions for reporting might not be satisfied, for instance due to the (lack of) subscribed data sets. The conditions for reporting, like conditions for logging is presented in section 2.1.5.

QueryLog

The client must be able to query the log. Two services exist for this purpose. The first service is named QueryLogByTime, and it specifies a time range. The system must return the log entries that have been logged between these two times. The second service is named QueryLogAfter, and it specifies a time and id. The system must return log entries that have been logged after the specified time and with an id greater than the id specified. The use of an id implies that the log entries must have a unique id for each log entry.

The intention for using time *and* id as parameters is due to the fact that multiple log entries can be inserted in the log at the same time (at a reasonably granularity for measuring time, for instance seconds). Query of the log just by time would potentially return multiple log entries not of interest. By use of the id, a precise starting point for the returned log entries is possible to define.

2.1.3 Mapping to web services

[61400-25-4] presents the mapping of the information model and the information exchange model to a specific protocol stack. Five actual mappings are presented in [61400-25-4] and the developers are free to choose the mapping they prefer. This thesis will use the SOAP based web services mapping.

In order to use the web services mapping it is worth considering a suitable environment for mapping (implementing) the system. The Windows Communication Foundation (WCF) has been found ideal for this purpose.

Windows Communication Foundation (WCF)

WCF is the new programming model from Microsoft which makes it possible to create services on Windows in accordance with SOA principles. WCF makes it possible to expose the native Common Language Runtime (CLR) as services and to consume other services as CLR types.

Productivity for the developer is increased because WCF makes it possible to focus on business logic rather than low level programming.

A WCF service can be viewed from following perspectives

- Business logic which implements the service to be provided.
- The hosting environment. The service has to exist in some context.
- One or multiple endpoints for the service where clients can connect in order to consume the service. An endpoint can be described by the ”abc” as will be presented.
- Exposing information that specifies how to communicate with the service and what to consume from the service. This is known as metadata exchange.

The business logic is related to the implementation of the logic representing the system.

Every service in WCF must be hosted in order to be available. It is possible to host a service with IIS, Windows Activation Service (WAS) or with a solution created by the developer also known as self hosting. When choosing the type of hosting, it must be considered that the system shall be mapped to web services and that it must support the publisher/subscriber pattern for reporting. All the listed possibilities for hosts support these features. Self hosting has been chosen in this thesis because it does not require IIS or WAS to be installed. In general, if available WAS should be preferred over IIS, because WAS is not limited to the HTTP protocol. However since web services use the HTTP, this is not of concern for the moment.

It must be considered with self hosting that it places more responsibility on the developer. For instance, the service must be manually launched before clients will be able to consume the service. With IIS and WAS, the service is launched automatically when the first client attempts to consume the service. An advantage for the self hosting is that it provides a familiar debugging environment because it is created as part of the application. This has been the main motivation for using self hosting in this thesis. Regarding self hosting it must be noted that it does not provide features which are built in with IIS and WAS such as robustness and recoverability. With this said, it must be mentioined that it will be possible to change the type of host in the future using the same implementation of the service. The proof of concept model can be created in a self hosting environment and in the future if it must be deployed in large scale, it could be hosted on WAS.

Each endpoint can be described with the ”abc” model.

- (A)ddress
- (B)inding
- (C)ontract

That is, an endpoint has an address (hence ”a”), it has a binding (”b”) and it has a contract (”c”). The ”abc” is visualized in figure 2.6. The address tells where the endpoint is hosted, the binding defines how the communication takes place and the contract defines the methods of the service that will be accessible via the endpoint. Every service is associated with a unique address. From the address it is possible to extract information about the location of the service and the transport protocol used to communicate with the service.

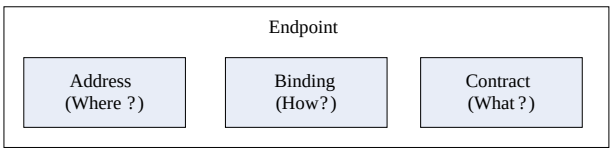

Figure 2.6: The abc of an endpoint

Binding is a WCF abstraction of communication and interaction related details. This includes the transport protocol used. WCF provides a variety of bindings and depending on the scenario the developer can choose the binding best suited. WSDualHTTPBinding supports callbacks which can be used to create a publisher/subscriber service for the reporting mechanism.

The contract defines, in platform neutral manner, which operations the service exposes. This type of contract is named service contract. Other types of contracts exists in WCF, such as the data contract which defines the data types that are exchanged with the service. WCF uses implicit data contracts for basic types such as integer and string. Explicit data contracts are not necessary in this thesis because only basic types such as integer and string are exchanged with the system.

In order for clients to know how and what to communicate with services WCF exposes metadata about this information. The metadata is communicated in platform neutral format WSDL/HTTP-GET. Two options for publishing metadata exist. First option is with HTTP-GET and second option is via a dedicated endpoint. With HTTP-GET, WCF is able to provide the metadata automatically. With the dedicated endpoint other protocols than HTTP is possible. However HTTP-GET is ideal for this thesis because the automatic approach is sufficient. By using the tool named svcutil.exe, a configuration file with details about address and binding for the service can be created. The tool also generates the service contract. Both the created files can be imported by WCF clients, which will then know the ”abc” of the system, that is, address, binding and contract. This will enable clients to consume services from the system. Platform neutral WSDL can also be generated from the metadata. This is achieved with the tool named disco.exe. Then any client capable of consuming web services will be able to use the system.

Service Oriented Architecture

In SOA, communication between services takes places with platform neutral messages. This supports interoperability across different platforms. Within the service the information can be converted to whatever format specified by the programming platform. The important part is that communication between services boundaries happens in a standardized fashion. Today this is realized by use of the soap/xml. The platform neutral messages are supported by the implicit (and explicit) data contracts in WCF. For instance the internal implementation of the IEC 61400-25 compliant system can created on the .NET platform and the client can be created on the JAVA platform. By using web services, they will be able to understand each other, because communication

between service boundaries is platform neutral.

2.1.4 WPP data generator

In order to test the system, data from a wind power plant is necessary. In an ideal environment, real wind power plants would have been used. However that approach is not possible in this thesis. An alternative is to have a WPP data generator which generates wind power plant specific data. Creation of such a generator depends on collaboration with people that has knowledge about wind power plants. This is not the case for the writer. As a consequence, the WPP data generator generates random data rather than wind power plant specific data. The system uses the WPP data generator in order to update values of the contents of the data model. The updating takes place at regular intervals and is the first step towards potential reporting and logging.

IEC 61400-25 does not define the communication between server and wind power plant. Polling would be a suitable approach, that is, the system explicitly asks the WPP data generator for data at regular intervals. One data attribute at a time, the system will be capable of retrieving values for all its data attributes, because it knows the contents of the wind power plant, due to use of the WPPCL file.

Length of the interval for polling can be decided based on the most critical information, which must be considered together with people that has knowledge about wind power plants.

2.1.5 Determining if reporting and logging must happen

After the system has initialized its data model according to the WPPCL file and started updating values for the data reporting and logging can occur. Whether it happens or not is determined by following

- The trigger option for the data attribute that has been updated
- Subscriptions for the data attribute

Besides, the value before and after the update might be necessary.

The process for determining if reporting and logging must occur is equivalent for both. Only difference between reporting and logging is outcome of the process. Reporting reports to the client (or buffers) while logging logs to persistent storage.

Trigger condition

First step for determining if reporting or logging must occur, is to know the trigger option of the data attribute which has been updated and if the condition for the particular trigger has been met after the update, that is

- Which trigger option does the data attribute have?
- Is the condition for this trigger option satisfied?

All data attributes have a field named trigger option as described earlier. The trigger can be dupd, dchg or qchg. However, the field can also be empty, meaning

that the data attribute has no trigger option. The consequence of not having a trigger option is that the data attribute can not cause reporting/logging to happen.

Determining if the condition for the trigger option is satisfied depends on the type of trigger

- Dupd. The condition for this trigger is satisfied immediately, because a data update occurred.
- Dchg. If the value after the update is different from the value before the update, then the condition for the trigger is satisfied.
- Qchg. Applies only to data attribute "q". If quality after the update is different from quality before the update, then the condition for the trigger is satisfied.

If the trigger condition has been met, this means that the first step towards reporting/logging has successfully been taken. Next step is to investigate if there is a subscription for the data attribute.

Subscription

The client expresses interest in certain data attributes by use of data sets. Data sets reference a group of data attributes, and the client subscribes to relevant data sets. The subscriptions for data sets happen at the level of (U)BRCB for reporting and LCB for logging. Each client has its own UBRCB, BRCB and LCB in order to manage its own subscriptions.

If it is determined that there is a subscription for the data attribute that was updated and whose trigger condition satisfied, then reporting/logging must occur. The reporting and logging is described in section 2.1.6 and 2.1.7, respectively.

2.1.6 Reporting

Reporting is the mechanism of the system reporting to the client, when a data attribute that the client has subscribed to satisfies the condition for reporting.

Publisher/subscriber

The publisher/subscriber pattern is used for reporting, rather than a polling approach. According to [61400-25-3] p.16 the server must be able to contact the client: "*Values can be reported to the client, following a publisher/subscriber reporting model (in the middle of the figure). The server is configured (locally or by means of a service) to transmit values spontaneously or periodically. The client receives messages (reports) whenever trigger conditions are met at the server.*". This concept is captured by the publisher/subscriber pattern.

The primary advantage for preferring publisher/subscriber over polling is that the client is delivered its reports immediately. With polling, the client has to ask the system for reports. Unless polling times are extremely short, publisher/subscriber will deliver the reports more timely than polling. A bonus of the publisher/subscriber approach is that no unnecessary load is placed on the network, system or client. The client does not have to ask continuously if

new reports have been generated. If timely retrieved reports must be obtained with polling, short polling intervals must be used, thus putting more load on system, client and network. The fact that multiple clients [1] must be able to use the system does not make it any better. However polling also has its advantage in terms of not causing troubles with security related obstacles such as firewalls. This is mentioned in the [61400-25-5] p. 58: *The reporting mechanisme specified has several benefits in complex communication environments with local and wide area networks involved including several layers of security obtaind via firewalls and routers*" [2].

No major drawback for using publisher/subscriber with reporting can be identified besides network related obstacles such as firewall denying the duplex communication necessary for spontaneous reports to be sent. Maybe the additional implementation required for maintaining the information about how to contact the clients, but that is a challenge in the implementation discipline rather than in the analysis discipline. By comparing the advantages and drawbacks for publisher/subscriber and polling, the publisher/subscriber approach is preferred because its advantages outweigh its disadvantages. Besides, no firewalls or similar are used between server and client in this thesis.

Multiple clients imply that reporting must occur separately for each connected client. As mentioned earlier, separate report control blocks are necessary for each client.

General reporting mechanism

Two types of reporting exist, that is, unbuffered and buffered reporting. Both types of reporting try to report to the client. The key difference between the two is in case of reporting failure, that is, if reports can not be delivered to the client. With unbuffered reporting, reports are simply lost. With buffered reporting, the reports are buffered until the client reconnects and retrieves its reports.

If reporting fails it is assumed that the network connection to the client is lost and the system registers that the connection to the particular client has been lost. The reason why the connection is lost, is not relevant. The important part is that it is not established. The system uses its knowledge about the connection state for a particular client when reporting must occur. If the connection is established, then reporting happens for both unbuffered reporting and buffered reporting. If the connection is not established, then the reports are discarded with unbuffered reporting, and buffered for later retrieval with buffered reporting. This saves the system from trying to send the report, when it knows that the connection is not established.

When the client reconnects to the system, the system will update its state and future attempts to report will be carried out rather than discarding or buffering the reports. This approach assumes that the client is the responsible part for reestablishing the connection to the system, when connection has been lost. Thus, the system is the passive part. If the client does not try to reconnect to the system, the connection will not be considered reestablished. This is a convenient approach because most of the time when connection is lost, it is

[1] According to Table 2 in [61400-25-3] p.18 it must be possible for multiple clients to receive information.

[2] Spelling errors are from the standard

caused by the client disconnecting from the system rather than problems with the network. It is not a convenient approach if the system is responsible for trying to detect if the clients are connected. The client disconnects, thus the system assumes this to be the case until the client explicitly reconnects.

Which information must the reports include? An approach would be to report the following information:

- Time and date where the reporting took place
- The data attribute that caused the reporting
- Value of the data attribute

In addition, buffered reporting must make use of a unique id for each report. This ensures that the client can know if it has received all reports or if some is missing. Besides, it can ensure that it retrieves the reports in chronological order.

In order to keep the system resources within safe boundaries, the system uses the parameters MinRequestTime and MaxRequestTime. Only within this window, reporting will be active. The window of time delimited by MinRequestTime and MaxRequestTime determines, how soon reporting will become active when the client has activated reporting and how long it will remain active. The client must explicitly activate reporting. When the system is running out of resources due to high number of reporting, one approach for addressing the challenge is to decrease the size of this window.

Unbuffered reporting

This is a best effort approach for reporting. The system tries to send the report, and if it fails, no further action will be taken, the report will be lost.

Buffered reporting

As is the case with unbuffered reporting, when conditions for sending a report to a given client are satisfied, the system generates the report and sends it to the client. If the report fails to make it to the client then the system will buffer the report until the connection is reestablished.

When the given client reconnects it will ask for buffered reports. If the client has any buffered reports, it will retrieve them. Retrieval of buffered reports will use a request-response approach rather than the publisher/subscriber approach used for ordinary reporting. The request-response approach has been chosen because it enables the client to determine the tempo for retrieval of buffered reports in order to do load balancing. That is, the client must able to say that it wants to retrieve the buffered reports one by one or in groups of ten or similar. Besides, the client will be able to delay a request if it experiences low system resources. This will allow devices with low capabilities in terms of memory and perfomance to use the system. If publisher/subscriber is chosen for the buffered reports, the client will have less control for the process. The argument for using publisher/subscriber regarding less load on network, client and system is not relavant for retrieval of buffered reports because it happens less frequent than the polling necessary for the general reporting mechanism.

A practical upper limit for the buffer must be considered in order to maintain the system resources in healthy condition. How the buffer is implemented is up to this thesis to determine. A first in first out (fifo) buffer is considered sufficient because the reports must be retrieved in the order they were generated. Buffered reporting, whether the reports are buffered or not, must use a unique id for each report. This ensures that the client can verify that no reports are missing and that reports are being retrieved chronologically.

When the client has retrieved its buffered reports they must be deleted from the buffer. However, the response part of the request-response for buffered reports may fail due to network related issues. In such case, the reports must not be deleted from the buffer, because the client has not actually received them. A solution to this challenge could be to make use of the unique id for each report. The client first learns the range of id's for its buffered reports. Then, one report at a time, it sequentially asks the system for a report with a particular id. When the system receives this request, it knows that the client must have received the report with id = n-1, thus the report can be safely deleted from the buffer. For instance if the client asks for buffered report with id = 15, the system knows that report with id = 14 must have been retrieved successfully by the client. Otherwise it would have requested report 14 again rather than requesting report 15. This approach also applies if reports are retrieved in groups of multiple reports rather than a single report at a time.

2.1.7 Logging

Logging is the mechanism of inserting entries in the log for later retrieval. It must be considered where the logged data is stored. Databases are widely used for this purpose in general and represent persistent storage. This means that log entries are unaffected of system shutdowns or crashes as opposed to buffered reports.

Retrieval of log entries happens by use of the two services QueryLogByTime and QueryLogAfter, which make make it possible to filter the log entries by time and id. This implies that every log entry must have a unique id.

2.1.8 Domain Model

With background in the analysis, objects in the domain have been identified along with their relationships. Figure 2.7 shows the domain model, which will inspire identifying objects for the design in section 3.

As can be seen on figure 2.7, the DataModel initializes itself by use of the WPPCL file. The DataModel contains the data model, but in order to keep the domain model simple, it has been drawn as one object. In order to view the contents of the DataModel, refer to figure 2.8.

After the DataModel has initialized itself, it uses the WPPDataGenerator in order to update values for its data attributes. While the update takes place, if it is determined that reporting or logging might occur, all the control blocks in the system are informed.

The number of control blocks in the system is three times the number of unique clients using the system (whether they are connected at the current moment or not) because each client has an UBRCB, BRCB and LCB. It is the responsibility of each control block to determine, whether reporting or logging

must occur, and if so, make it happen. Each control block uses a Subscription object in order to manage its subscriptions. The subscription object references one or multiple DataSet objects depending on its current state of subscriptions. Each data set references zero or multiple data attributes in the data model.

2.1.9 Use cases

The use cases express the requirements to the system, seen from the perspective of the client. In order to identify use cases, the groupings of the services in the information exchange model in section 2.1.2 will be used as inspiration. The reason for the correspondence between the information exchange model and the use cases is considered because both concepts capture which functionality the client must be able to consume from the system.

It is important to note that the use cases do not express requirements to the complete system (or the thesis, for that matter). Only requirements that the client expects from the system (seen as a black box entity) is captured by the use cases.

Considering the information exchange model at the level of each single service, is too low level, in terms of abstraction, for writing use cases. This is where the groupings of the services will be useful. Following use cases have been identified for the system.

Use case 1: RetrieveDataModelContents

Primary actor: Client
Stakeholders and Interests: Client wants to retrieve the contents of the data model in the system.
Preconditions: Client has an established connection to the system, and is associated with the system.
Postconditions: Client has retrieved the contents of the data model.
Main Success Scenario: 1. Client specifies data of interest. This can for instance be retrieval of all logical devices in the system 2. The system returns the contents of the data model.
Open Issues: None.

Use case 2: SetSubscription

Primary actor: Client
Stakeholders and Interests: Client wants to subscribe or unsubscribe to a given data set for a given control block.
Preconditions: Client has an established connection to the system, and is associated with the system.
Postconditions: Client has subscribed or unsubscribed for a given data set for the given control block.
Main Success Scenario: 1. Client specifies the data set of interest and chooses to add the subscription for a particular control block (UBRCB, BRCB or LCB). 2. The system adds a subscription for the data set to the given control block.
Alternative Flow: 1. The client specifies to remove subscription for a data set in a particular control block. 2. The system removes the subscription for the data set in the given control block.

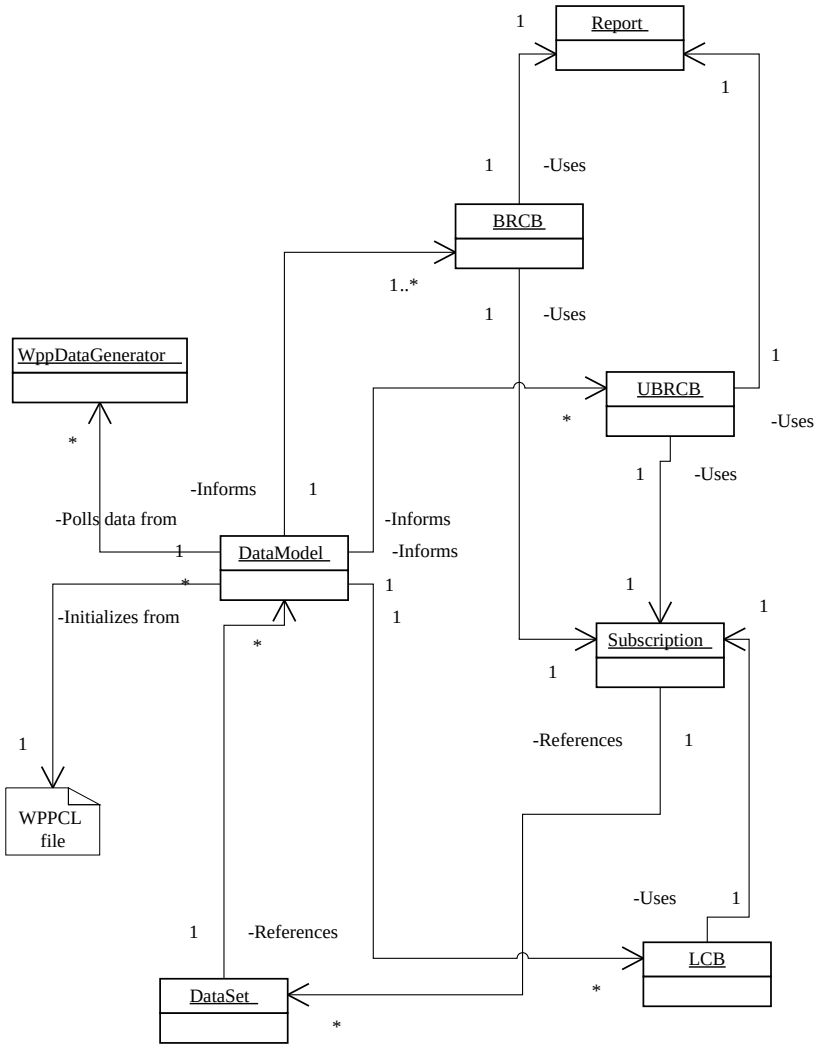

Figure 2.7: Domain model

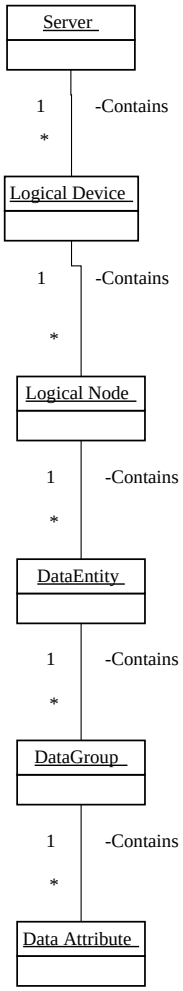

Figure 2.8: Data model

Open Issues: None.

Use case 3: GetSubscriptions

Primary actor: Client
Stakeholders and Interests: Client wants to retrieve subscribed data sets for a given control block.
Preconditions: Client has an established connection to the system, and is associated with the system.
Postconditions: Client retrieves a list of subscribed data sets for the given control block.
Main Success Scenario: 1. The client wants to retrieve a list of subscriptions for a given control block. 2. The system returns the data sets that the particular control block (UBRCB, BRCB or LCB) has subscriptions for.
Open Issues: None.

Use case 4: GetDataSetValues

Primary actor: Client
Stakeholders and Interests: Client wants to retrieve the data attributes that are referenced by a given data set.
Preconditions: Client has an established connection to the system, and is associated with the system.
Postconditions: Client has retrieved the data attributes that are referenced by the given data set.
Main Success Scenario: 1. Client specifies the data set of interest. 2. The system returns the data attributes that are referenced by the given data set.
Open Issues: None.

Use case 5: Reporting

Primary actor: Client
Stakeholders and Interests: Client wants to receive reports spontaneously.
Preconditions: Client has an established connection to the system, and is associated with the system.
Postconditions: Client has received reports spontaneously.
Main Success Scenario: 1. Client subscribes to relevant data sets. 2. The system reports to the client spontaneously if conditions for reporting are satisfied.
Alternative Flow: If the client is not connected to the system when the report is sent, the report must be either discarded (unbuffered reporting) or buffered (buffered reporting). The client must be able to retrieve the buffered reports when it reconnects by use of request-response approach.
Open Issues: None.

Use case 6: QueryLog

Primary actor: Client
Stakeholders and Interests: Client wants to retrieve the log entries.
Preconditions: Client has an established connection to the system, and is associated with the system.

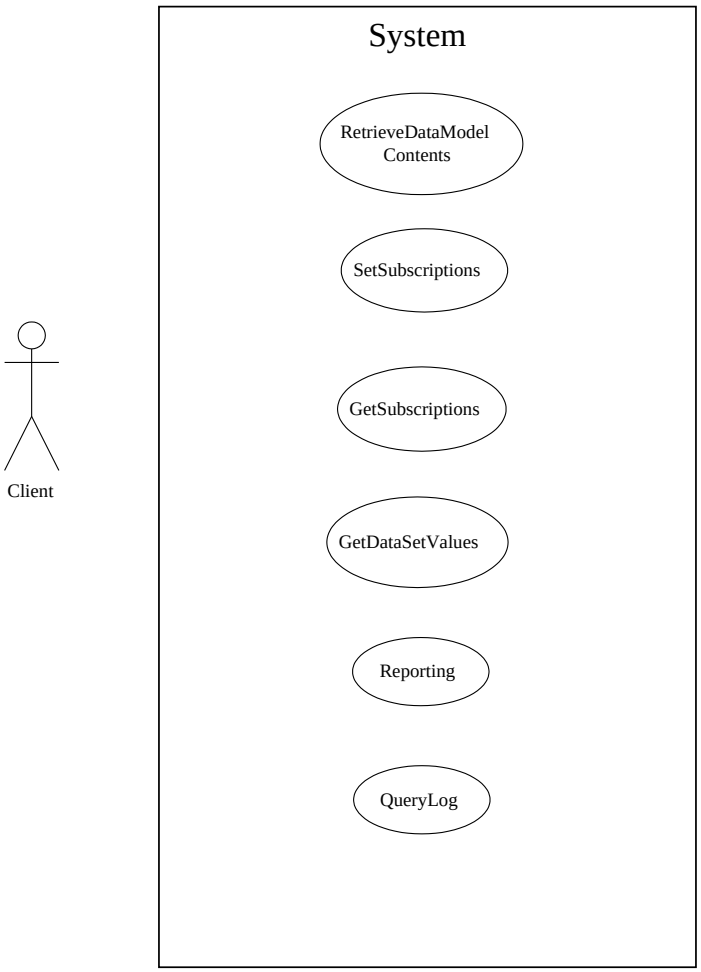

Figure 2.9: Use case diagram

Postconditions: Client has retrieved log entries.

Main Success Scenario: 1. Client specifies a time range that it wants to retrieve log entries from. 2. The system returns the log entries that have been logged in the specified time range.

Alternative flows: 1. Client specifies a time and a id, after which it wants to retrieve log entries. 2. The system returns log entries that have been logged after the specified time *and* id.

Open Issues: None.

2.1.10 Use case diagram

The use case diagram in figure 2.9 provides a visual overview for the identified use cases.

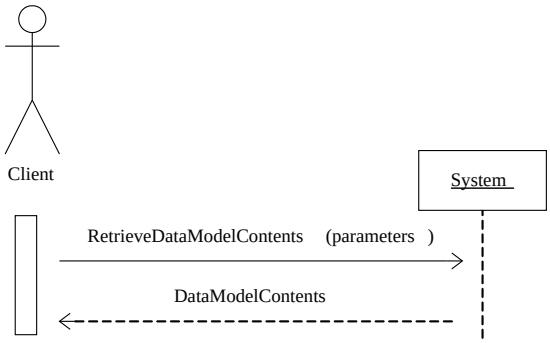

Figure 2.10: Ssd: RetrieveDataModelContents

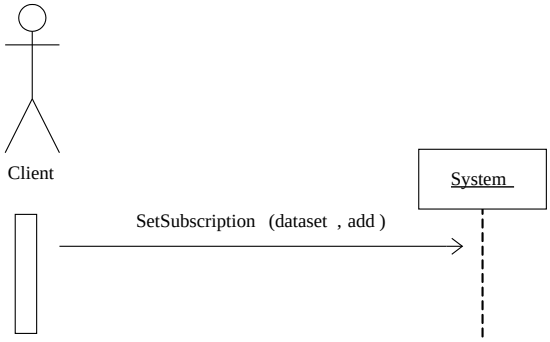

Figure 2.11: Ssd: SetSubscription

2.1.11 System Sequence Diagrams

The interaction between client and system as defined in the use cases is visualized by use of system sequence diagrams.

RetrieveDataModelContents

The client specifies the data of interest and requests it. For instance the client may want to retrieve the names of all logical nodes within a given logical device. The system returns the data of interest. The interaction can be seen in figure 2.10.

SetSubscription

The client wants to either add a subscription or to remove subscription. As mentioned earlier, subscriptions are at the level of control blocks (UBRCB, BRCB, LCB). The client specifies which data set it wants to add or remove subscription for, and the method is carried out for the given control block. The interaction can be seen in figure 2.11.

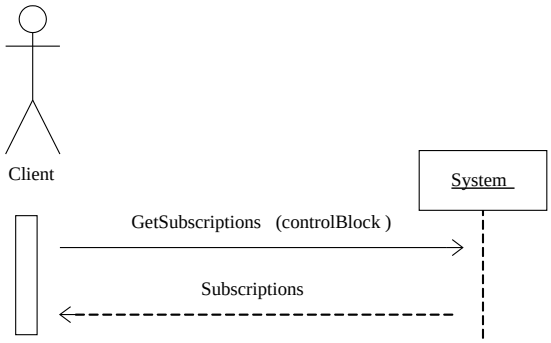

Figure 2.12: Ssd: GetSubscriptions

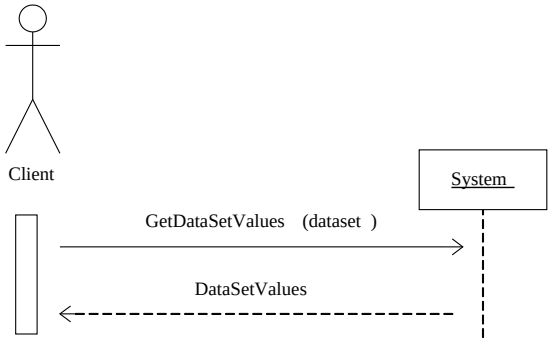

Figure 2.13: Ssd: GetDataSetValues

GetSubscriptions

The client wants to retrieve a list of subscriptions for a given control block. The system returns subscriptions for the given control block in terms of data sets, as can be seen in figure 2.12.

GetDataSetValues

Each data set references a group of data attributes. The client wants to retrieve the list of data attributes referenced by a given data set. As can be seen in figure 2.13, the system returns the referenced data attributes.

Reporting

The client wants to receive reports from the system spontaneously. In order to achieve this, the client must add subscriptions (for reporting related data sets) via the SetSubscription service, and then wait for the system to send back reports. However reports are not guaranteed to happen, because the conditions for reporting may not be satisfied. The interaction for reporting can be seen in figure 2.14.

System sequence diagram for retrieval of buffered reports is not shown.

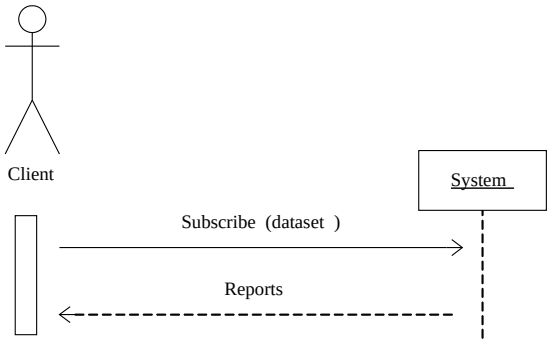

Figure 2.14: Ssd: Reporting

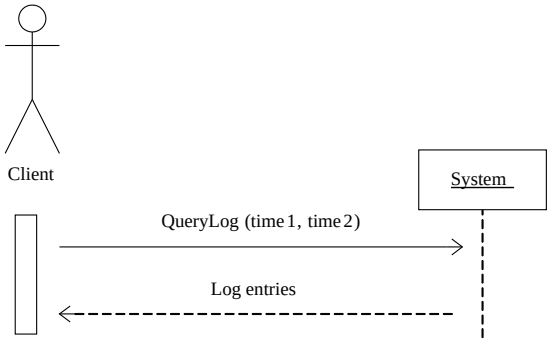

Figure 2.15: Ssd: QueryLog

QueryLog

The client must be able to query the log by either a time range, or a time and id. Figure 2.15 shows the client querying the log by a time range. The system returns the log entries that have been logged within the time range specified.

2.2 WPPCL file

The purpose of the WPPCL file is to serve as specification for the contents of the data model that a given wind power plant supports. When selecting a format for WPPCL, it must be considered that the data model is a hierarchical data structure. Expressing a hierarchical data structure can be achieved by xml. In data model terms this means that the xml file consists of the server element at the top level and the data attribute element at the bottom level.

A convenient feature of xml is that it is pure text based, thus it will be platform neutral. Being text based, it will also be straightforward to send the WPPCL file across different network related obstacles such as firewalls, for instance by sending it as attachment with an email.

Who does have the responsibility of creating the WPPCL file? Although no standards addresses this question, this thesis proposes, that it must be the

responsibility of the wind power plant manufacturer to create it as part of other documentation for the wind power plant. However in this thesis no such WPPCL file is available from a wind power plant manufacturer, which is why it must be created by this thesis.

Use of the WPPCL file is related to the subject "Configuration" in [Andreas & Baris] where following three options for configuring the data model are described

- Design time
- Deployment time
- On the fly

In this thesis, with configuration, contents of the data model is meant. In [Andreas & Baris] other subjects such as length of interval for polling the wind power plant is also considered with configuration. This is not the case in this thesis. Configuring at design time is a static approach where the data model contents from the wind power plant is known at design time and is considered not likely to change in the short future. This approach does not make use of the WPPCL file. Rather the system will only be able to model one type of wind power plant with only a single specification for contents of the data model. The contents of the data model are simply written as part of the business logic for the system and compiled as part of the application. This strategy implies that if contents of the data model in the wind power plant changes, then the contents of the data model in the system must be modified and recompiled in order to reflect the new contents. Changes in the contents of the data model in the wind power plant can be caused by a new type of wind power plant or as a result of software update in an existing wind power plant. The reason for changes in contents of the data model is outside scope for this thesis to investigate. Note that the system will have a downtime equal to the time it takes to modify, recompile and test the modified system before it is able to continue servicing its clients.

Configuring the data model at deployment time means that the initial business logic in the system is independent of the type of wind power plant which the system has to model (as mentioned earlier, only IEC 61400-25 compatible wind power plants are considered). At deployment time (initial startup) the system reads the WPPCL file and configures (initializes) its data model contents in accordance with the file. Seen from the flexibility perspective, configuration at deploy time is an improvement compared to configuration at design time, because the system is not tied to a specific instance of contents in the data model.

Moving one step further towards flexibility, configuration on the fly is met. This method, like the deployment time approach, makes use of the WPPCL file. The key difference is that configuring on the fly checks the WPPCL file at regular intervals for changes and reflects this in the data model contents. This means that the system does not need to be restarted caused by changing contents of the data model. However the high flexibility comes at a cost. For instance if the system has clients subscribed for reporting for a given data attribute and suddenly this data attribute disappears from the system as a result of change in the WPPCL file, how should the system cope with this? Behavior must be defined in such a situation.

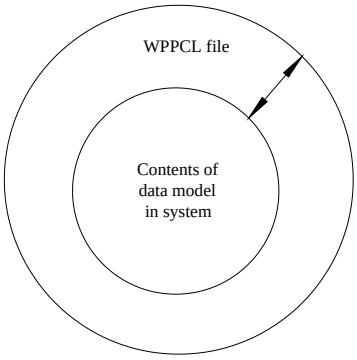

Figure 2.16: Contents of data model in the system is a subset of the contents of the WPPCL file

Based on properties for the three types of configuration this thesis will use the configuration at deployment time approach. It is considered the most balanced approach, because the system has the same business logic for using various wind power plants, thus it will be able to model all IEC 61400-25 compatible wind power plants. Besides potential undefined behavior with "on the fly" approach is avoided, that is, no data attribute from the system will be added or removed while clients are being served.

The use of configuration at deployment time is in accordance with the approach proposed by [Andreas & Baris].

2.3 WPPCL Editor

The WPPCL file includes all data elements that a given wind power plant supports. If some of those elements are not of interest for a given system then this can be reflected in the WPPCL file by simply deleting elements not of interest by use of a text editor. However this approach is not without concerns. First, manual editing assumes that the person doing the job must be familiar with xml. This may not always be the case. Even if it is the case, then manual editing is error prone, for instance due to typing errors. Second, when deleting elements not of interest, what happens in the future, if the deleted elements become relevant for the system due to changing client requirements? An original copy of the WPPCL must be stored to handle this scenario.

In order to avoid deleting elements not of interest, a field is introduced for each data attributes in the WPPCL file, indicating whether it is enabled of disabled. The system will only include enabled data attributes when initializing contents of its data model. However this still leaves the error prone manual editing as an unanswered issue. Which is why a tool for editing the WPPCL file is proposed by this thesis. The tool is named WPPCL Editor.

Within the boundaries of the WPPCL file it is possible to customize the contents of the data model in the system by use of the WPPCL Editor as can be seen in figure 2.16

Considering the fact that the data model is hierarchical, the WPPCL Editor must provide a graphical user interface where all contents of the data model

as specified in the WPPCL file is visualized. For the different elements it shall be possible to point and click in order to enable or disable a given element. It must be possible to distinguish between enabled and disabled elements by use of different coloring. When the editing is over, saving the configuration must be possible. WPPCL Editor must reflect the changes to the WPPCL file. Next time the system initializes its data model, changes in the WPPCL file will have effect.

2.4 Client

The client is considered the actor using the system. It is the responsibility of the client to use the services exposed by the system's information exchange model. The client must be able to carry out following operations

- Subscribe/unsubscribe to data sets for reporting
- Enable/disable data sets for logging
- Retrieve data attributes referenced by a given data set
- Retrieve contents of the data model
- Receive spontaneous reports from the system and retrieve buffered reports
- Query the log

2.4.1 Reporting

It must be possible to subscribe to data sets both for buffered reporting and unbuffered reporting. Subscriptions for reporting is separate for unbuffered and buffered reporting and consequently they are configured independently. The process of subscribing must be done by viewing a list of available data sets for reporting and then selecting the relevant data set where subscription is desired. This process is repeated until subscriptions are completed. Next step is to activate reporting.

When data sets are no longer of interest it must be possible to unsubscribe from these. This process is carried out by selecting the data set that is currently subscribed to but no longer of interest and unsubscribing them. The process is repeated until all data sets no longer of interest have been unsubscribed. It must be possible to deactivate reporting (separately for unbuffered and buffered reporting), when reporting is not supposed to happen. Subscriptions are not affected by activating or deactivating reporting.

2.4.2 Logging

Logging from the client's perspective relates to two tasks. The first task is to configure the subscriptions for the logging. Managing subscriptions for logging is similar to reporting. The second task is related to retrieving the contents of the log, which must be possible by filtering on time and entry id.

2.4.3 SCADA

The client needs to have an interface to humans. The interface must make it possible to configure and use the reporting and logging. Humans are not considered as actors in the system, because they interact with the system through the client.

The interface is named SCADA in wind power plant context and is not within scope of IEC 61400-25. Nonetheless it is part of this thesis to ensure and test the monitoring system from the operator's perspective. Therefore it is up to this thesis to create an intuitive and user friendly design for the SCADA.

Command line and graphical user interfaces are normally the options considered, when creating user interfaces. Due to the hierarchical structure of the data model in the system, it is convenient to visualize it as a tree. This proposes that a graphical user interface must be used, which will be the case in this thesis.

2.5 Requirements

With background in the analysis, the requirements for the thesis can be defined.

A system that supports the reporting and logging aspects of IEC 61400-25 must be designed, implemented, mapped and tested. In particular, the system must meet the following requirements

- The system must contain an information model as specified in section 2.1.1 with separate control blocks (UBRCB, BRCB and LCB) for each client. The structure for the data model must be hierarchical with the server element in the top down to the data attribute in the bottom. Data sets must be created by the system according to the rules in [61400-25-2].
- The system must expose services as specified in section 2.1.2.
- The system must be mapped into the web services protocol. WCF is used for mapping the system, following the guidelines for SOA. Thus, no platform specific data must be exchanged between service boundaries. WSDualHttpBinding provided in WCF is used for the duplex communication required by the spontaneous publisher/subscriber approach for reporting.
- The system must initialize contents of its data model by use of the WPPCL file.
- After initialization the system must poll for data from the WPP data generator. Interaction between system and WPP data generator must happen by means of polling mechanism.
- The system must monitor its regular data updates and determine if reporting or logging shall occur.
- The system must support both unbuffered and buffered reporting. The reporting mechanism must make use of publisher/subscriber pattern. When connection between system and client is not established, reports must be discarded (unbuffered reporting) or buffered (buffered reporting). Buffered reports must have a unique report id for each report. The buffered reports must be possible to be retrieved by request-response approach. The

buffered reports must be removed from the buffer when retrieved by the client.

- The system must use MinRequestTime and MaxRequestTime in order to balance load caused by reporting.

- Persistent storage must be used for inserting log entries and it must be possible to query the log at a later time.

In addition to the system, the WPPCL file must be created. The format for the file is xml. The WPPCL file must be possible to configure by use of the WPPCL Editor, which must be designed and constructed. The WPPCL editor must provide a graphical user interface visualizing the contents of the WPPCL file. It must be possible to enable and disable data attributes. Coloring must be used in order to distinguish between enabled and disabled elements. When editing is finished, the changes must be saved to the WPPCL file.

The WPP data generator must be created, in order for the system to use it for polling data. In lack of real wind power plants, this is the chosen approach. The WPP data generator, despite its name, will not produce realistic wind power plant specific data. In order to achieve this, collaboration with people that has knowledge about wind power plants is necessary.

A client that is capable of using the services exposed by the information exchange model including reporting and logging must be designed and constructed. The client must provide a graphical user interface representing a SCADA in order to provide an interface for human interaction.

2.6 Conclusion

This chapter has analyzed relevant topics for creating an IEC 61400-25 compliant system with focus on reporting and logging. Details in the standard not relevant, such as controlling of wind power plants, is out of scope and has been left out.

The system consists of an information model with separate control blocks (UBRCB, BRCB and LCB) for each client. This ensures, that each client can configure its own reporting and logging. Data sets must be created by the system by using predefined rules in accordance with [61400-25-2]. The information model must have a hierarchical data model with the server element in the top down to the data attribute element in the bottom. Data attributes are possible to address with a unique path in the form

```
LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute
```

The system consists of an information exchange model, which exposes services necessary for configuring and using reporting and logging, along with general use of the system.

The system is mapped to the web services mapping by use of WCF. The duplex communication necessary for sending spontaneous reports is provided by the WSDualHttpBinding in WCF. Platform specific data are kept within the system, only platform neutral messages are exchanged with the system. This ensures interoperability between platforms.

Both types of reporting as defined in IEC 61400-25 are supported, that is, unbuffered and buffered reporting. The reporting mechanism makes use of the publisher/subscriber pattern, which ensures that the client receives reports spontaneously. In case of not established connection between client and system, the reports will be buffered (buffered reporting) or discarded (unbuffered reporting). Buffered reporting uses unique id's for each report in order for the client to determine if it has received all reports. The client must retrieve all its buffered reports when reconnecting to the system. Retrieval of buffered reports happens by use of request-response rather than the publisher/subscriber approach used for ordinary reporting. This provides the client more control with the process, thus the client will be possible to retrieve buffered reports according to its resources. This approach will enable devices with low resources such as handheld devices to use the system.

The system uses the terms MinRequestTime and MaxRequestTime in order to be able to perform load balancing. If too much reporting occurs, it can decrease size of the window, where reporting remains active.

Regarding logging, it happens to persistent data storage for later retrieval. In accordance with IEC 61400-25, the log entries can be queried by the two services QueryLogByTime and QueryLogAfter.

Furthermore the analysis has dealt with topics out of scope for IEC 61400-25 such as configuration of the contents of the data model in the system and how values of the data must be updated by use of a WPP data generator. Configuration of data model contents makes use of the WPPCL file, an xml file, which must be created by the wind power plant manufacturer, however in this case it must be created by the thesis. It must be possible to configure the contents of the WPPCL file by use of WPPCL Editor, which will ensure against error prone manual text based editing. Changes to the WPPCL file is reflected in the contents of the system's data model at next initialization. A polling approach has been chosen for the system retrieving data from the WPP data generator. WPP data generator is a random data generator rather than a realistic WPP data generator. To improve the WPP data generator, wind power plant competent people must be cooperated with. An alternative to the WPP data generator is to use a real wind power plant. However, this is left open for future work.

Updating the values of the data model contents by use of WPP data generator is the first step towards potential reporting and logging. The process for determining if reporting and logging must happen is similar. It depends on the trigger option for the data attribute that was updated, and it depends on the given subscriptions, in terms of data sets, for the data attribute. If the condition for the trigger option has been met after the update, and the data attribute is referenced by data sets that subscriptions exist for, then reporting and logging occurs.

The client interacts with the system by use of the services that the system exposes. This includes managing the subscriptions for reporting and logging. The client has a graphical user interface, representing a SCADA, which enables an operator to use the system. The SCADA visualizes the contents of the data model by use of a tree structure, utilizing the fact that the data model is hierarchical.

Chapter 3 will propose approaches for design in order to meet the requirements defined in this chapter.

Chapter 3

Design

With background in the analysis, this chapter will design the different parts of the thesis. In particular, design for the following will be proposed

- The IEC 61400-25 compliant system. The system must meet its external and internal requirements. External requirements are considered according to the use cases. Internal requirements includes the system initializing its data model by use of the WPPCL file and updating its data model by use of the WPP data generator at regular intervals. Reporting and logging must be considered as a potential consequence of the updating.
- The WPPCL file and, in order to customize it, WPPCL Editor.
- A client including SCADA that is capable of using the system.

The IEC 61400-25 compliant system will be designed in section 3.1. The WPPCL file and WPPCL Editor will be designed in section 3.2 and 3.3, respectively. Section 3.4 designs a client with SCADA that is capable of consuming the services exposed by the system. Section 3.5 concludes the design chapter.

3.1 IEC 61400-25 compliant system

In order to design the IEC 61400-25 system, necessary objects to fulfill the requirements will be identified by use of interaction diagrams. Section 3.1.1 will create interaction diagrams for the use cases. Section 3.1.2 presents interaction diagrams for initialization related tasks, such as the system initializing contents of its data model by use of the WPPCL file. Section 3.1.3 presents interaction diagrams for the updating mechanism in the system, which can cause reporting and logging to happen. The system uses the WPP data generator for the updating. Reporting and logging are the subjects of sections 3.1.4 and 3.1.5, respectively.

Identified objects and their relationships will be presented in the design class diagram in section 3.1.6.

3.1.1 Use case realizations

The use case realizations will be created with reference to the use cases identified earlier.

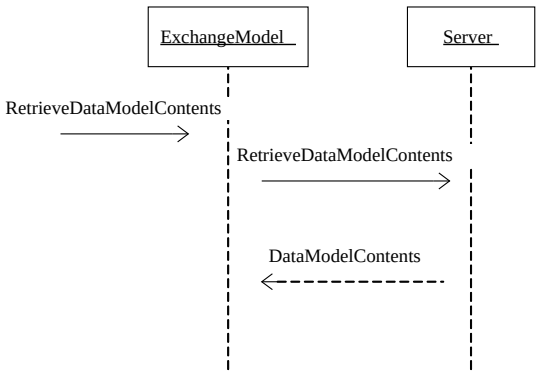

Figure 3.1: Ucr: RetrieveDataModelContents

Entry point to the system is the ExchangeModel object (abbreviation for information exchange model). ExchangeModel implements the interface IExchangeModel which defines the services that must be exposed by the system. IExchangeModel can be considered as the contract for the system. IExchangeModel defines a callback contract of type IReportCallbackContract. This callback contract defines a service named OnCallBack(). The detailed explanation of this is related to the mapping and WCF, and will be dealt with in section 4. For now it will be sufficient to mention, that every client which wants to consume services of the system, must have a service named OnCallBack. This is the service that the system will use in order to get in touch with the client when reporting must happen.

Rather than doing all the work, ExchangeModel delegates most of the work to relevant objects, representing Information Experts in the system. To the client, ExchangeModel represents the system. Hence, the ExchangeModel can be considered as facade controller in pattern terms.

RetrieveDataModelContents

When contents of the data model are requested by the client, ExchangeModel delegates the work to the Server object, which is the top level of the data model. The Server object is considered Information Expert, in pattern terms, regarding the data model. Consequently, all requests regarding the data model is issued to the Server object. For instance if the client wants to retrieve all the data attributes within a given location (logical device, logical node, data entity, data group) it provides the ExchangeModel with the parameters. ExchangeModel delegates the task to Server, retrieves the contents, and returns them to the client. The interaction diagram for retrieving data model contents can be seen in figure 3.1.

SetSubscription

ExchangeModel exposes control block objects to the client, that is, one UBRCB, one BRCB and one LCB per client.

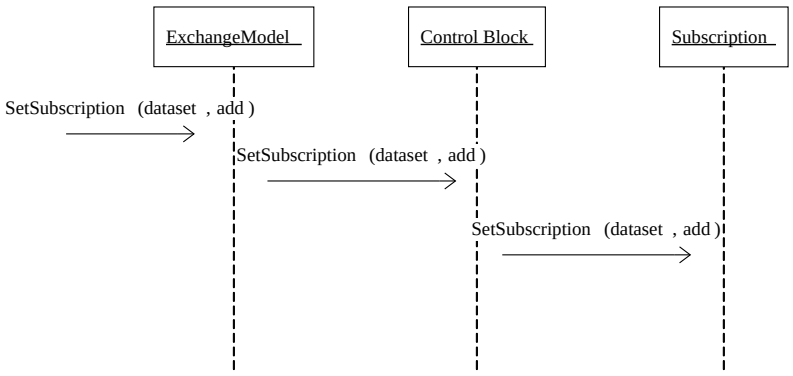

Figure 3.2: Ucr: SetSubscription

By use of the control block objects, the client is able to manage its subscriptions for reporting and logging. For at given control block, the client specifies a data set and if it must be subscribed or unsubscribed to. ExchangeModel delegates the task to the given control block and supplies it with the name of the data set and whether subscription must be added or removed. It is the responsibility of the control block objects to store the subscriptions. For this purpose, each control block object uses a Subscription object. The subscriptions are at the level of data sets, thus the Subscription object operates with data sets. The control blocks use the object DataSetManager for data set related tasks, such as retrieving a reporting or logging related data set. The interaction diagram for setting subscription for a given control block can be seen in figure 3.2.

DataSetManager is considered Information Expert regarding data sets. It has the responsibility of knowing the rules for creating the data sets and to actually create them. DataSetManager must know the Server object, which contains the data model, in order to create the data attribute references. It must search the data model in the Server object for data attributes that match the rules of the data sets. Each time a match is found, a reference to the data attribute is created and added to the given data set. References have the format

```
LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute
```

DataSetManager is responsible for knowing which of the data sets that are reporting related and which of them that are logging related. This information is used by the control blocks, that is, UBRCB and BRCB ask DataSetManager for reporting related data sets, and LCB asks DataSetManager for logging related data sets. DataSetManager must know, which data attributes the data sets reference. This will be used in the process for determining if reporting and logging must happen.

GetSubscriptions

The client wants to get a list of its subscriptions, in terms of data sets, for a given control block. ExchangeModel delegates the task to the given control block object, which in turn asks its Subscription object. The subscriptions,

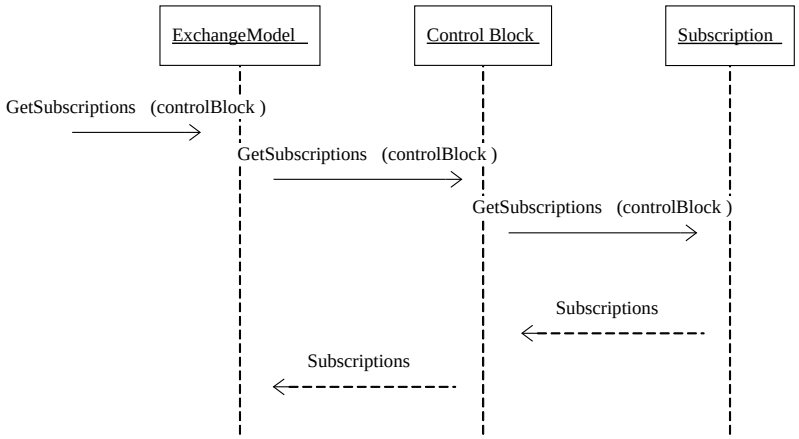

Figure 3.3: Ucr: GetSubscriptions

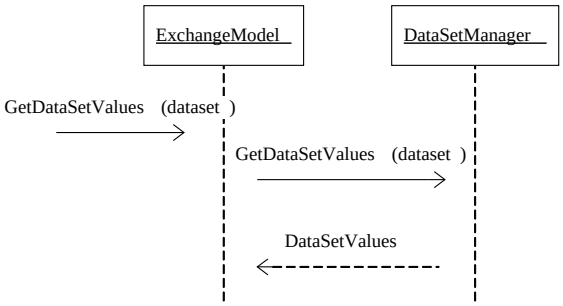

Figure 3.4: Ucr: GetDataSetValues

represented as data sets, are returned to the client, as can be seen in figure 3.3.

GetDataSetValues

The client wants to get a list of data attributes that are referenced by a given data set. ExchangeModel uses the object DataSetManager to answer the question and returns the list of data attributes to the client, as can be seen in figure 3.4. As mentioned before each data attribute reference has the format

```
LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute
```

Reporting

Potential reporting starts with updating of the data model in the system. This use case realization is related to the updating mechanism of the server, which can be seen in section 3.1.3. For the actual reporting, refer to section 3.1.4.

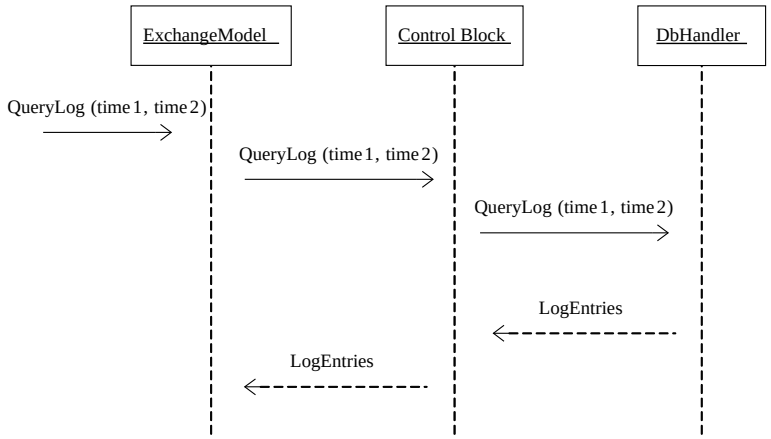

Figure 3.5: Ucr: QueryLog

QueryLog

When log entries must be retrieved, ExchangeModel asks LCB, which in turn knows the DbHandler object. DbHandler is responsible for all persistent data storage related tasks and it is the only entry point to the persistent data storage. Thus, DbHandler can be considered as a Facade.

DbHandler retrieves log entries from the data storage and returns them to LCB. The number of retrieved log entries depend on the filter, which the client has specified by (time1 and time2) or (time, id). LCB returns the results to ExchangeModel, which is able to service the client. The use case realization for retrieving log entries can be seen in figure 3.5.

3.1.2 Initialization

Initially the system has no contents in its data model. When the system is turned on, it reads the WPPCL file and creates contents of the data model in accordance with it. This ensures that the system reflects the contents of the data model of the wind power plant that it models. After initialization, the system starts regular updates of the values of the data model contents by the updating mechanism, as will be described in section 3.1.3. From this point on, the system is ready to service the clients. Each client that connects to the system gets its own unique set of control blocks. Each control block that is created, in turn creates its own Subscription object, in order to maintain subscriptions. The Subscription object operates with objects of type DataSet.

Data Model

When the system starts up the first time it initializes contents of its data model. The Server object is container for the data model and as mentioned before, it is considered Information Expert regarding the data model. This includes the initialization of the data model. Naturally, the Server object does the initialization task by itself. The initialization happens by reading the WPPCL file and creat-

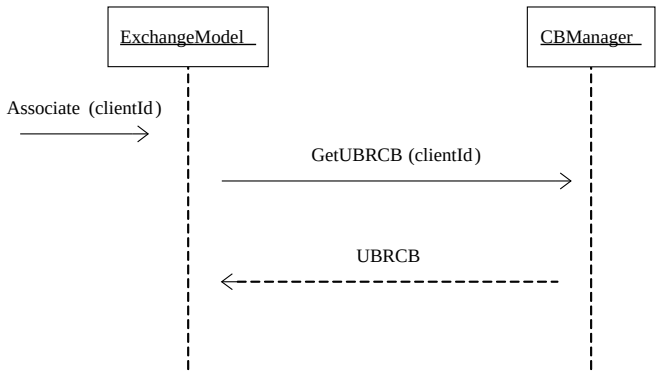

Figure 3.6: GetControlBlocks

ing the corresponding logical devices, logical nodes, data entities, data groups and data attributes. Only data attributes that are enabled in the WPPCL file are included in the data model. There is no interaction diagram showing the initialization of the data model because the Server object is doing the job itself.

Control Blocks

When a client connects to the system it must be allocated its own set of control blocks. If it is first time the client connects to the system, new control blocks must be created and associated with the client. On the other hand, if the client has connected to the system before, then it can use its already allocated set of control blocks. CBManager (abbreviation for ControlBlockManager) is responsible for creating and managing control blocks. Association with the system is required before any interaction with the system can take place. The client provides its unique id while associating with the system. This is the id which CBManager uses to determine if it is a new client or a returning client. The interaction of the association and the allocation of control blocks can be seen in figure 3.6.

As can be seen in figure 3.6, the association message is retrieved by the ExchangeModel object. ExchangeModel determines, by looking in the client id, if the client is a valid user in order to be associated with the system. Although not a subject in this thesis, this is a simple approach for access control. If association is successful, the client must get is own set of control blocks in order to use reporting and logging. ExchangeModel sends the message GetUBRCB(clientid) to CBManager, which sends back an UBRCB object to ExchangeModel. It is the responsibility of CBManager to either create a new UBRCB or to use an existing UBRCB in case the client is known.

CBManager is considered as Information Expert regarding knowledge about the clients that is using the system, whether they are connected or disconnected. Every time a new client connect to the system, its existence is recorded in CBManager in terms of control blocks. Each control block object has a field named id. This corresponds to the client id. Thus, it is possible for the CBManager to see, if a client is a new client or if it is a returning client by looking in its collection for control blocks with id's that match the id of the client. CB-

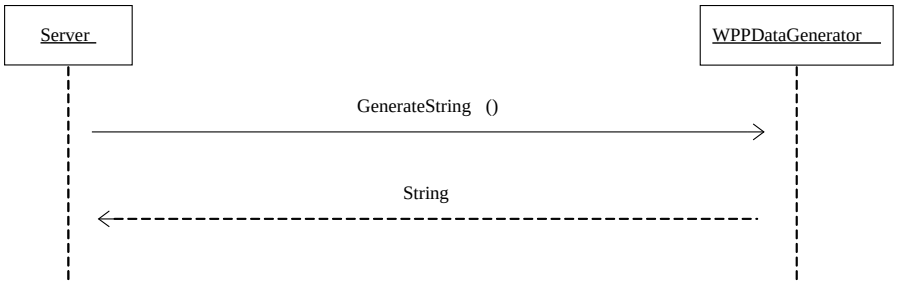

Figure 3.7: Server gets wpp data from WPP data generator

Manager is a Singleton, only one CBManager exists in the system. This is the chosen approach, because it makes good sense to have central management of the information regarding clients and control blocks.

Data Sets

Subscriptions are carried out at the level of data sets. Data sets must be created after the data model in the system has been initialized, because the contents of the data model are not known in advance. Thus, outcome of the creation of data sets can be different, depending on the contents of the data model, even when data set rules are identical.

DataSetManager is the responsible object for creating the data sets and handling data set related tasks such as knowing if a given data attribute is referenced by a given data set.

DataSetManager could actually create the data sets directly from the WPPCL file, because it has the same contents as the system. However, the WPPCL file also includes disabled elements, so DataSetManager would have to handle this in order to avoid creating references for data attributes that are disabled. In order to obtain low coupling and not having to handling enabled and disabled elements, the WPPCL file is chosen not to be used by DataSetManager.

No interaction diagram for creating data sets is present, because DataSetManager does the job by itself.

3.1.3 Updating

The updating takes place at regular intervals and the purpose is to get new data for the contents of the data model from the WPP data generator. A potential outcome of the updating is that reporting or logging must occur. The Server is Information Expert when it comes to updating its data since it holds the data model. It contacts the WPP data generator in order to get current data for all its data attributes, as can be seen in figure 3.7. The updating takes place one data attribute at a time. The Server object loops through all its data attributes in the data model and for each data attribute it asks WPP data generator for a value. As mentioned before, WPP data generator generates a random value, rather than realistic wind power plant data. It generates a random value that has a type corresponding to the type of data attribute and returns the value. The type of the data attribute is specified by the Server object. For instance, in

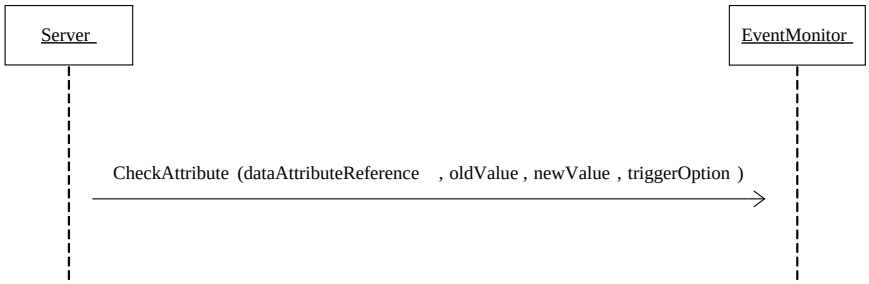

Figure 3.8: Server contacts EventMonitor

figure 3.7 the type of the data attribute is a string, and the WPP data generator generates and returns a string.

A possible outcome of the updating is that reporting or logging must occur. In order to investigate this, the Server delegates the task to the EventMonitor object. Every time a data attribute has been updated, the Server object contacts the EventMonitor. It passes along the following information

- Reference to data attribute that has been updated
- Old value for the data attribute
- New value for the data attribute
- Trigger option for the data attribute

Interaction between Server and EventMonitor can be seen in figure 3.8. With the parameters provided from the Server, EventMonitor is able to determine if the condition for a trigger is satisfied. As mentioned earlier three types of triggers exist: Dupd, dchg and qchg. Conditions for the triggers are

- dupd: This trigger is satisfied immediately, because the update takes place.
- dchg: If the new value is different from the old value, this trigger is satisfied
- qchg: Applies only to data attribute of type q (quality). The trigger is satisfied, if the new quality is different from the old quality.

If the trigger option for the current data attribute is dupd, then the trigger condition is satisfied immediately. This is due to the fact that the Server only informs the EventMonitor when a data update takes place. The dchg trigger depends on the values after and before the update. If the values are different, then the condition for dchg has been satisfied. It must be noted that from the system's perspective there is no difference between the two triggers dchg and qchg. This is because a data attribute of type quality is treated like all other values such as strings or numbers. If change has happened then a dchg has happened. In other words qchg is a part of dchg.

From the perspective of EventMonitor, it does not matter whether dupd or dchg was satisfied. The important part is that a trigger condition has been satisfied, thus further action must be taken. If no trigger condition was satisfied, there is no more work for the EventMonitor to do in relation to the current data

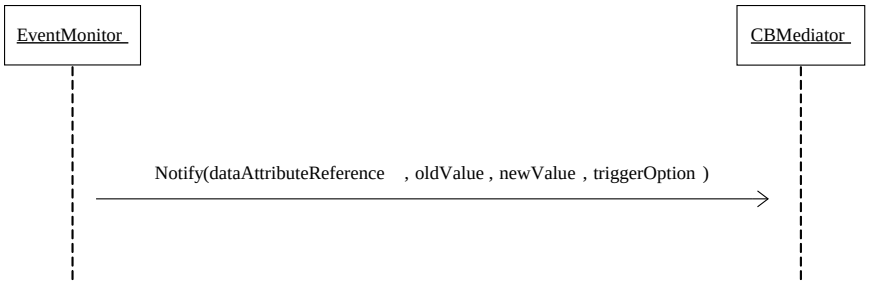

Figure 3.9: EventMonitor informs CBMediator

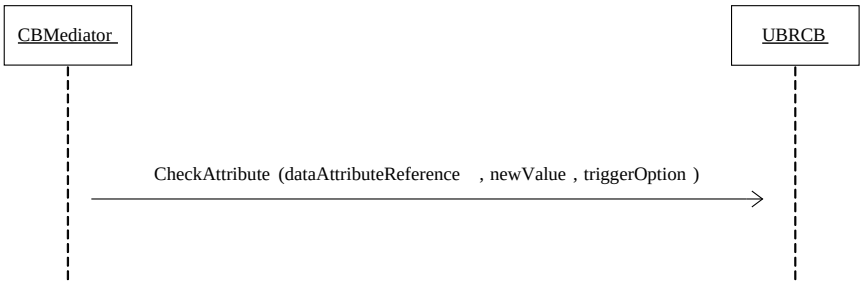

Figure 3.10: CBMediator informs UBRCB

attribute. It waits to be contacted by the Server next time a data attribute gets updated. If a trigger condition is satisfied, it the responsibility of EventMonitor to inform the object CBMediator (abbreviation for ControlBlockMediator). Interaction between EventMonitor and CBMediator can be seen in figure 3.9.

The responsibility of CBMediator is to inform every control block in the system. Thus, CBMediator is considered as a Mediator, in pattern terms. CBMediator knows all the control blocks in the system indirectly because it knows CBManager. As mentioned before, CBManager knows all control blocks in the system. Each control block is responsible for determining if it must report/log. Control blocks determine this by looking in their subscriptions. Interaction between CBMediator and control block, in this example UBRCB, is illustrated in figure 3.10.

When CBMediator has informed all control blocks, there is no more work for it to do related to the current data attribute. What happens after this depends on the state of subscriptions in the control blocks.

The objective of each control block is to determine if it has a subscription for the data attribute. This can be done by looking in its Subscription object to retrieve its subscriptions and then for each subscribed data set use the DataSetManager to investigate if the data attribute reference is included in the data set. If it is the case then reporting or logging must occur depending on the type of control block. Up to this point the logic for reporting and logging has been identical. This is why common tasks for reporting and logging have been chosen to be represented as an abstract object named ControlBlock. The interaction diagram in figure 3.11 shows how the control block object determines if

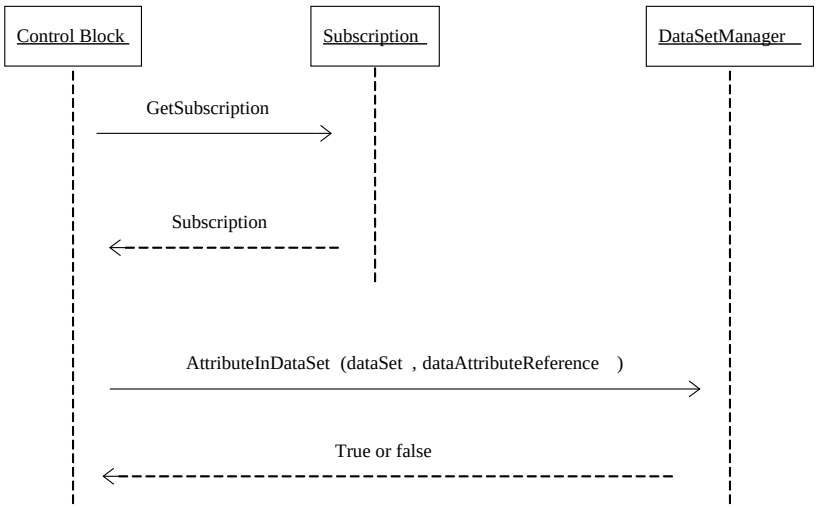

Figure 3.11: Control block (UBRCB, BRCB or LCB) determines if reporting/logging must occur

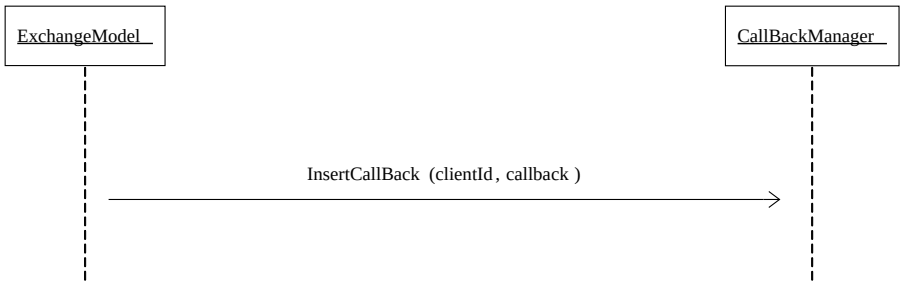

Figure 3.12: Storing contact details for the connected client

reporting/logging must occur.

3.1.4 Reporting

When the client associates with the system, ExchangeModel uses the object CallBackManager to store contact details for the client. This ensures that the system is able to send reports to the client. CallBackManager is Information Expert regarding contact details for the clients. The interaction between ExchangeModel and CallBackManager can be seen in figure 3.12.

UBRCB and BRCB both makes use of the Report object in order to do the reporting. The Report object contacts the CallBackManager in order to get contact information for a given client, as can be seen in figure 3.13.

Figure 3.14 shows an UBRCB using the Report object for reporting. The interaction is similar for BRCB.

If the Report object detects a failure with sending the report, it first contacts the CallBackManager and sets the connection state for the particular client to

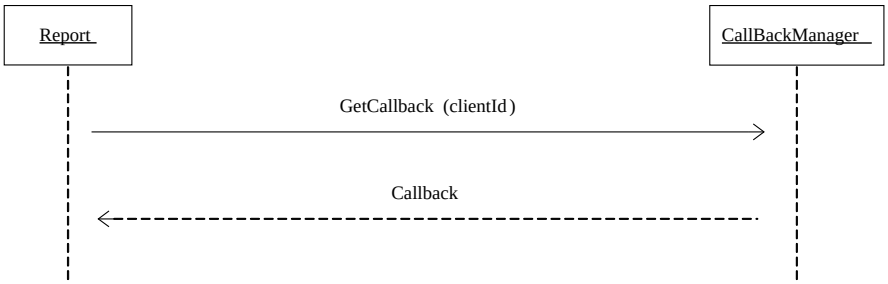

Figure 3.13: Getting callback for the client

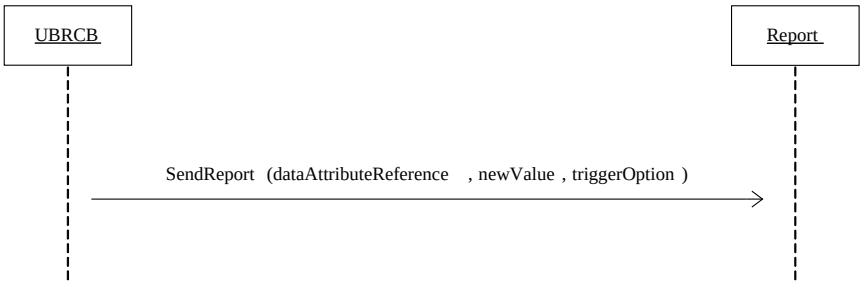

Figure 3.14: Reporting

not established. This ensures that future attempts to report to the client does not waste time by trying to send reports. Next, the Report object sends back a message to the RCB that contacted it (UBRCB/BRCB) and informs it that reporting has failed. As response to a reporting failure, UBRCB buffers the report, while UBRCB simply does nothing, and the report is lost.

The reporting related control blocks, UBRCB and BRCB both makes use of the Report object, and as such an abstraction level between the ControlBlock object and the UBRCB and BRCB would be ideal. This abstract level is represented as the RCB object, and it encapsulates reporting related tasks, such as knowledge about the Report object. Placing this information in the ControlBlock would also have worked, but it would have violated the "is-a" rule for LCB which also is a control block. LCB shall not have any knowledge about reporting.

The RCB knows MinRequestTime and MaxRequestTime. When a RCB is activated by the client, a timer is started and the activation of the RCB is delayed until MinRequestTime has elapsed. When MaxRequestTime has elapsed, the RCB is deactivated.

When a client connects to the system, it asks the ExchangeModel if it has any buffered reports since last time it was connected. ExchangeModel delegates the request to the BRCB for the client. If buffered reports exist , the client is responsible for retrieving the buffered reports. Thus, this is a normal request-response approach rather than the publisher/subscriber approach for ordinary reporting.

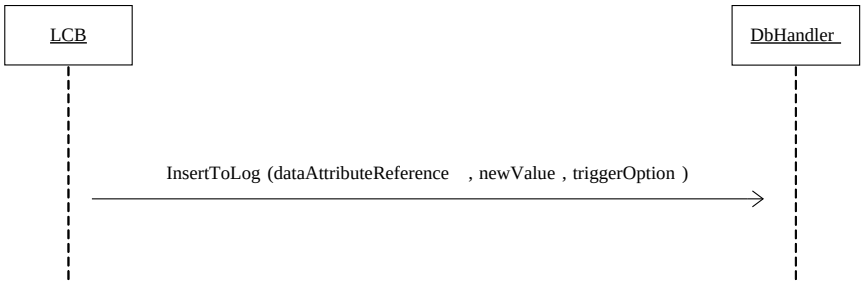

Figure 3.15: Logging

3.1.5 Logging

The LCB is responsible for logging, and as menitioned before, LCB is a specialization of the ControlBlock object. When conditions for logging are satisfied, LCB uses DbHandler in order to insert log entries in the persistent storage. Interaction diagram for logging can be seen in figure 3.15.

Data storage

The data storage is used for logging. The type of storage is a persistent solution, a widely used approach for logging of various data. A possible approach would be to log the following information every time logging takes place

- Unique id for the log entry
- Time and date where the logging took place
- Name of the data attribute that caused the logging
- The value for the data attribute

The above information could be stored in a single table in a relational database. The reason for having the unique id field is that multiple log entries can be inserted in the log at the same time (at granularity level of seconds). By use of the id field it will be possible to differentiate between log entries.

A single table holds the entries and the layout can be seen in table 3.1. As can be seen in table 3.1, log entries can be described by four fields. ID,

Field Name	*Data Type*
ID	AutoNumber
Timestamp	Data/Time
ObjectReference	Text
CurrentValue	Text

Table 3.1: Table for storing log entries

Timestamp, Objectreference and CurrentValue. The ID field is the primary key and is associated with an AutoNumber. This means that every time a new entry is inserted in the log a new row is created with a new ID number. The ID number

is an integer and grows incrementally from one to infinity. The objectreference is the unique path for the data attribute, which has been presented earlier.

The method for inserting entries in the log is not specified by the IEC 61400-25 which is why at proprietary method is needed. The method is named InsertEntryInLog for convenience.

When log entries must be retrieved, the two services QueryLogByTime and QueryLogAfter are used. The service QueryLogByTime takes two time/date arguments. Entries within the boundary of these two time/date arguments are returned by the service. The QueryLogAfter service also takes two arguments. The first is a time/date argument and the second is an ID argument, that is, the primary key in the log entries.

3.1.6 Design Class Diagram

The objects discovered while creating the interaction diagrams will be presented in the design class diagram. To provide a clear overview, the design class diagram has been split into three parts. The design class diagrams have been created manually in order to keep information that is too low level out of the diagrams in order to obtain a clear overview.

Figure 3.16 shows the contract (interface) IExchangeModel which is implemented by ExchangeModel (in figure 3.17). IExchangeModel has defined a callback of type IReportCallbackContract that every client must implement in order to consume services from the system. This will be described in more details in section 4.

Figure 3.17 shows the part of the objects that is responsible for the general working of the system, that is, everything besides reporting and logging. The ExchangeModel object is responsible for exposing the services defined by IExchangeModel.

When a client connects to the system, it must be associated with the system before it can consume the services. This could be a candidate for access control. The ExchangeModel models this by simply looking at the client id, and determining if it must be granted or denied access. If successful association, ExchangeModel must allocate ControlBlock objects for the given client and make sure it has access to the contents of the data model. For allocating ControlBlock objects, ExchangeModel delegates the job to the CBManager object, and provides it with the client id. The Server object is a Singleton, due to the fact that only one data model can exist in the system. Thus, multiple ExchangeModel objects use the same Server object.

ExchangeModel also makes sure that contact details about the client is stored by use of the object CallBackManager. Later, the Report object will make use of the CallBackManager to retrieve contact details about a given client in order to do the reporting. The CBManager is also a Singleton, and knows all ControlBlock objects in the system. Each control block has an id field that corresponds to the client id. Based on the client id, CBManager either creates new control blocks for the client, or delivers existing control blocks. The ControlBlock object is an abstract object that groups together control block related tasks for reporting and logging, such as managing subscriptions.

After the Server object has initialized its data model by use of the WPPCL file, it starts to update the values of the contents of its data model by use of the WPPDataGenerator object. For each data attribute that is updated, the Server

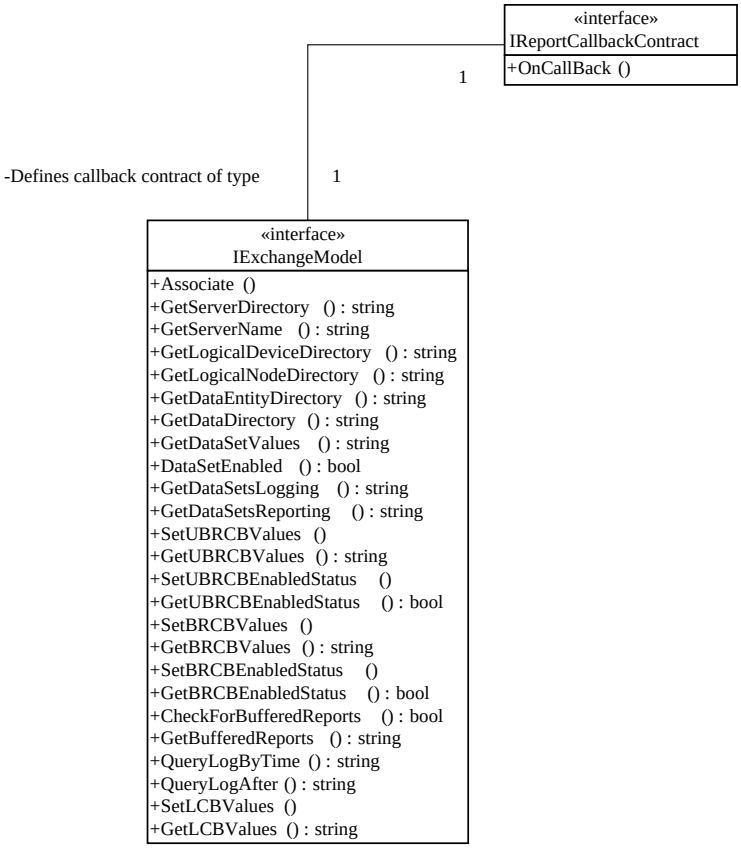

Figure 3.16: Design Class Diagram, part one

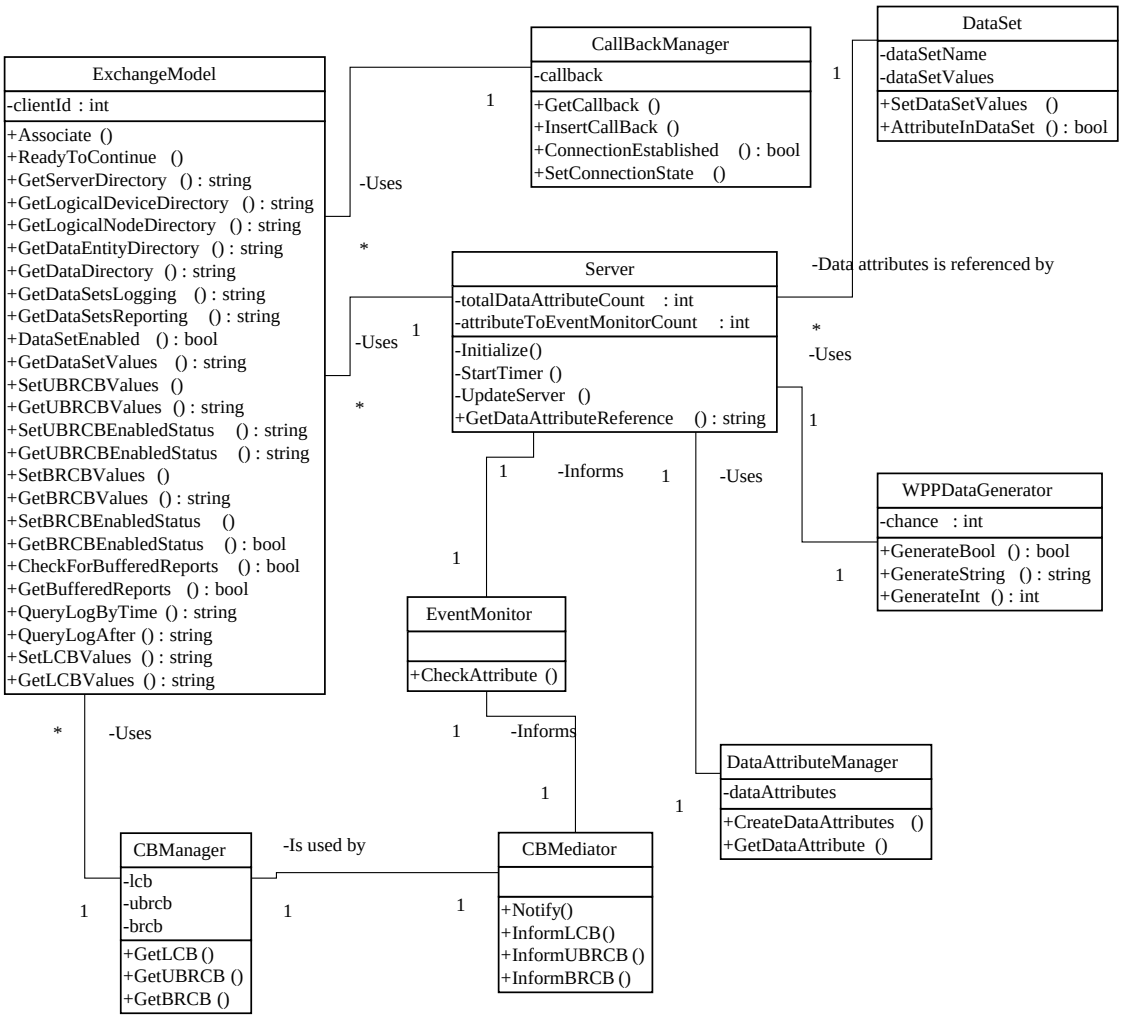

Figure 3.17: Design Class Diagram, part two

informs the EventMonitor object. The responsibility of the EventMonitor is to check, if the trigger condition for the data attribute has been satisfied. If so, EventMonitor informs the CBMediator object, which in turn informs all the ControlBlock objects in the system in order for them to determine if reporting and logging must occur. CBMediator indirectly knows about all ControlBlocks in the system because it knows CBManager.

Figure 3.18 shows the reporting and logging related part of the objects. The object CB (abbreviation for control block) is an abstract class, which contains methods common for reporting and logging. It uses a DataSetManager object in order to know about data sets and subscribe to them. It uses a Subscription object to maintain its subscriptions. The DataSet object contains a name and a list of data attribute references. Objects of type DataSet is created by DataSetManager, according to the predefined rules for the data sets.

The abstract class RCB is a specialization of CB in the sense that is adds reporting relevant information. RCB uses the Report object for reporting. BRCB creates a unique id for each report and it has a buffer for buffering reports.

LCB uses the object DbHandler for the logging to persistent storage and retrieval of log entries. DbHandler is considered a facade.

3.2 WPPCL file

The WPPCL file must be in xml and must reflect the hierarchy of the data model. The data attributes must have a field, where it is possible to see if the data attribute is enabled or disabled. Following design for the WPPCL file will be used

```
<server servername = "">
  <ld ldname="">
    <ln lnName="">
      <data dataName="" commonDataClass="">
        <dataGroup dataGroupName="">
          <dataAttribute dataAttributeName="" enabled="false" />
        </dataGroup>
        <dataGroup dataGroupName="">
          <dataAttribute dataAttributeName="" enabled="true" />
          .....
```

3.3 WPPCL Editor

The graphical user interface for the WPPCL Editor can be seen in figure 3.19. At the left side, contents of the data model in the WPPCL file is presented in a tree structure. Data attributes use coloring. Green means that a data attribute is enabled and red means that it is disabled.

Customizing the contents of the data model is straightforward. It is done by simply clicking a data attribute and then either clicking the "Enable" button or the "Disable" button.

When customizing is complete, changes are saved by clicking the "Save" button. If changes must be discarded, the WPPCL Editor is just closed. Saving will only take place by explicitly pressing the "Save" button.

3.4 Client

The client is supposed to use the services exposed by the system. This means that reporting and logging must be possible to use from the client, besides general use of the system such as retrieval of data set values. The client has a graphical user interface, representing a SCADA, in order to receive commands from humans using the system and to visualize contents of the reporting and logging.

Figure 3.20 shows the graphical user interface for the client. Starting from the top left, the contents of the data model in the system is presented as a tree. The presentation of the data model as a tree is not used for reporting or logging

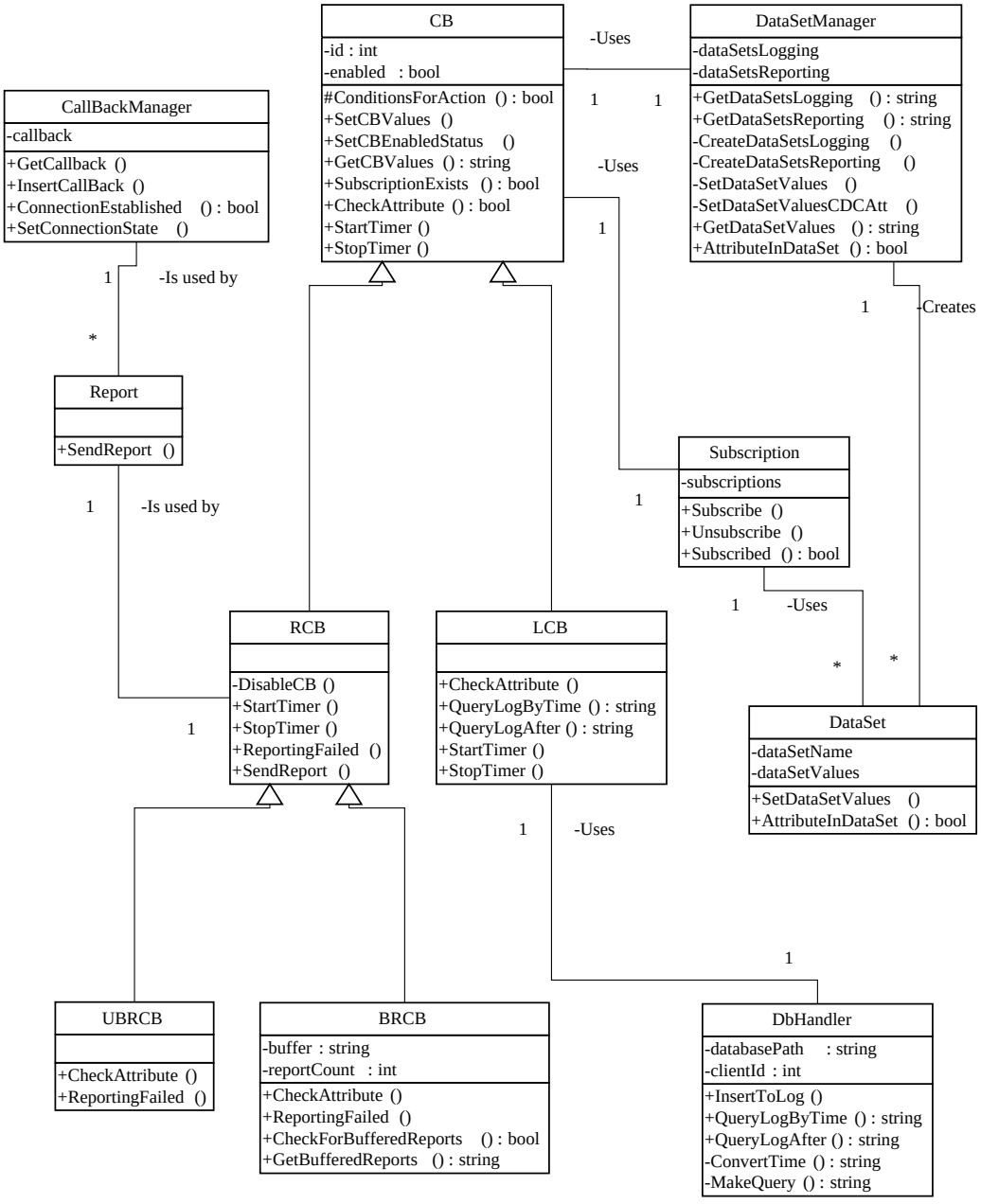

Figure 3.18: Design Class Diagram, part three

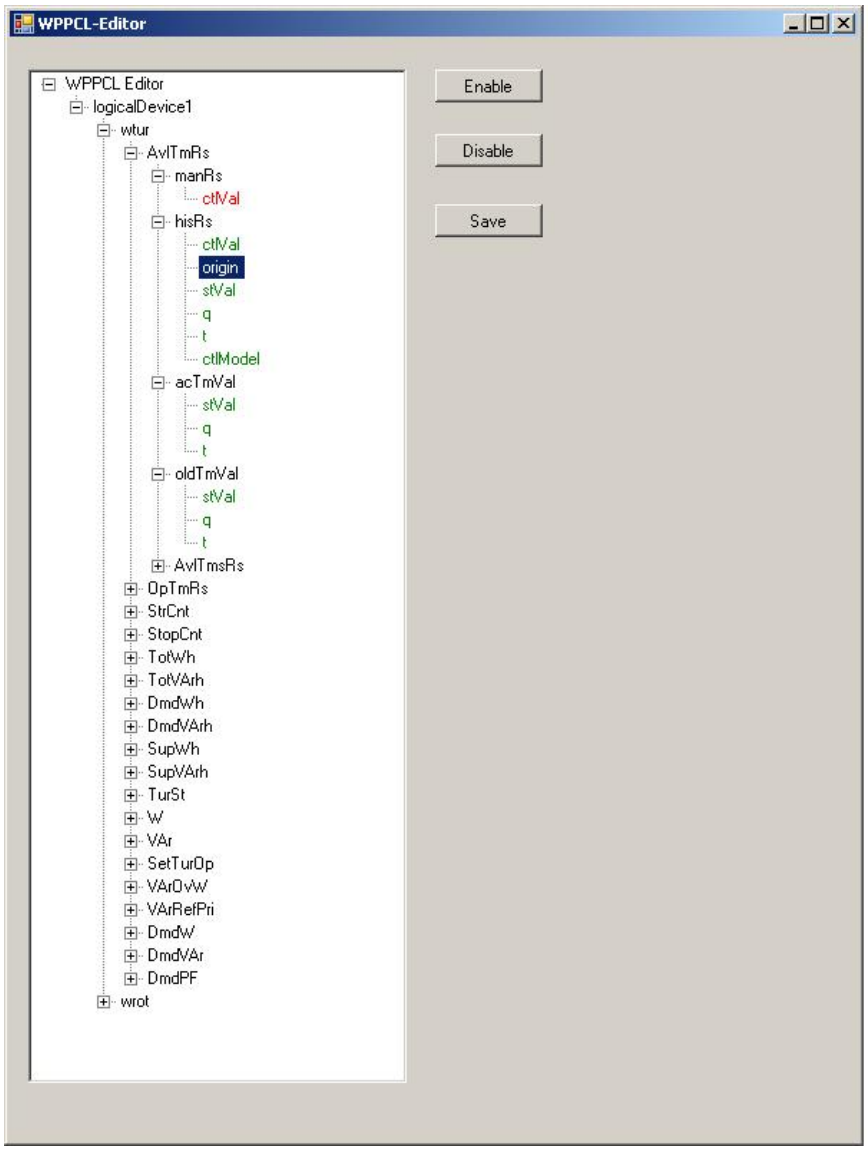

Figure 3.19: User interface for WPPCL Editor

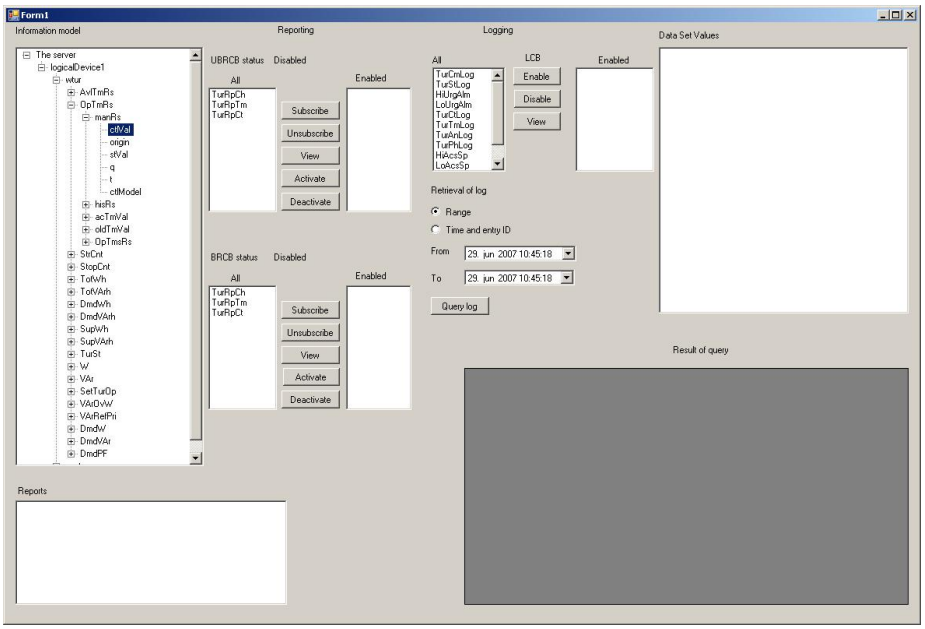

Figure 3.20: User interface for client

by the client. Its purpose is to provide an visual overview of the contents of the data model.

Below the data model, the area for reporting is seen. In this area, the client writes reports that it receives (spontaneous) or retrieves (buffered reports, request-response) from the system.

Moving on to the top in the middle, configuration of unbuffered reporting is possible. Data sets for reporting are shown, and the client is capable of managing its subscriptions by the buttons ”Subscribe” and ”Unsubscribe”. By clicking the ”View” button, data attribute references for the given data set is retrieved. Data set contents are shown at the very right side.

The client must activate reporting by clicking the ”activate” button. In the top, a field shows the status for unbuffered reporting, that is, if it is enabled or disabled. The status may not change to enabled immediately due to the MinRequestTime used by the system. When reporting is activated by the system (after MinRequestTime has elapsed), the system informs the client in order for it to update its state for reporting. The client can choose to deactivate reporting it self, or the system will do it after MaxRequestTime has elapsed. When the system disables unbuffered reporting, it informs the client, in order for it to reflect this to the graphical user interface.

Below unbuffered reporting, buffered reporting can be configured. The mechanism is the same as for unbuffered reporting.

At the top right, configuration of logging is possible. The process for managing subscriptions and viewing data set contents are identical to the reporting. Logging however is always enabled, thus there is no buttons for activating/deactivating it. It is possible to query the log by selecting either ”By Range” or by ”Time and entry ID”. The parameters are set below. The time parameter

is selected with a user friendly graphical calendar. Results of the query (if any) are written in the bottom right side.

3.5 Conclusion

This chapter has proposed an object oriented design for meeting the requirements to the system. Besides this, the WPPCL file, WPPCL Editor and client have been designed.

Regarding the system, relevant objects have been identified and their interaction has been shown by use of use case realizations. Furthermore the internal workings of the system such as polling the WPP data generator and detecting potential reporting and logging has been described and visualized by use of interaction diagrams.

The ExchangeModel object is a facade controller pattern that represents the information exchange model. It is the only point of interaction for the client. ExchangeModel delegates most of the work to other objects rather than doing the tasks by itself. The delegation of tasks corresponds to the Information Experts in the system.

It is the responsibility of the Server object to initialize the contents of its data model by reading the WPPCL file. After initialization, the Server uses WPPDataGenerator object to update values for each data attribute in its data model.

Common reporting and logging related tasks, such as managing subscriptions, has been grouped in the abstract object CB. Common reporting related tasks have been grouped in the abstract object RCB. CBManager makes sure that each client is allocated its own set of control blocks for reporting and logging when connecting to the system for the first time. The control blocks are stored by CBManager and when the client reconnects to the system, it uses its existing control blocks.

Although not an objective of the thesis, a simple access control to the system has been suggested at the entry point to the system, that is, ExchangeModel. It is an all or nothing approach, that is, either access is granted or denied to the complete system. In reality, a finer grained approach would be used, for instance by specifying the parts of the data model, which the client shall have access to. Access control can be a future topic for the system.

Objects responsible for unbuffered and buffered reporting, UBRCB and BRCB, respectively, both uses the Report object to do the actual reporting because they are both of type RCB. If reporting fails due to not established connection between client and system, UBRCB discards the reports, while BRCB buffers them. BRCB generates a unique id for each report. The Report object uses CallBackManager in order to retrieve contact details about the client that must be reported to. ExchangeModel makes sure, that client details are recorded in CallBackManager when clients associate.

Regarding logging, the responsible object is the LCB. For tasks related to the persistent data storage, LCB uses the object DbHandler, which is a facade, because it is the only point of interaction to the persistent storage.

All the identified objects and their relationships to other objects have been shown in a design class diagram.

The client has been designed with focus on the SCADA, which has been chosen to be graphical in order to visualize the contents of the data model as a tree. The design of the client makes it possible to consume the services exposed by the system, including reporting and logging.

The WPPCL Editor has been designed with intuitive user interaction in focus. The WPPCL Editor makes it possible to enable and disable data attributes in the WPPCL file.

Next chapter will implement, or in IEC 61400-25 terms, map, the system into web services by use of WCF and implement a client that is capable of consuming the services exposed by the system. It will also create the WPPCL file and the WPPCL Editor. The wind power plant data generator will be created as a random data generator.

Chapter 4

Implementation

The purpose of this chapter is to map the system into web services. The chapter will describe key aspects of the different software classes. Section 4.1 describes the implementation of the IEC 61400-25 compliant system. It is also the purpose of the chapter to implement a client with a graphical user interface that is capable of using the system. Section 4.2 describes the implementation of the client.

WCF is used as the framework for implementing both the system and the client [1]. However, because the chosen mapping is web services, they could be implemented on different platforms because only platform neutral data types are exchanged between system and client. For instance, the client could be implemented by use of JAVA.

The WPPCL file is created in section 4.3 and the WPPCL Editor capable of editing the WPPCL file is implemented in section 4.4.

Conclusion of the chapter can be found in section 4.5.

4.1 IEC 61400-25 compliant system

The system is mapped to web services by using the WSDualHttpBinding binding (WS is abbreviation for Web Service). The use of WSDualHttpBinding rather than WSHttpBinding, which is also a web service mapping, is dictated by the publisher/subscriber pattern used for reporting, that is, WSDualHttpBinding provides duplex communication on top of the HTTP protocol used for web services.

4.1.1 General system

Entry point for the system is the ExchangeModel object. This object implements the IExchangeModel interface which defines the services exposed by the

[1] Visual C sharp 2005 Express Edition has been used as IDE for construction of the system and client. Microsoft Windows Software Development Kit (SDK) for Windows Vista and .NET Framework 3.0 Runtime Components must be installed on the computer. After installation, reference to the SDK in terms of System.ServiceModel must be added. This is achieved from the IDE by r-clicking on the project and selecting ”add reference” and then localizing the System.ServiceModel.

system. In IEC 61400-25 terms, the IExchangeModel defines the Information Exchange Model.

As an example of the contents of IExchangeModel, consider the contract for the service GetServerDirectory below

```
[OperationContract]
string[] GetServerDirectory();
```

The [OperationContract] parameter specifies that the method will be exposed by the system. Without this, it would be a normal interface declaration. Note that the service must return an array of strings with the logical devices in the system. As mentioned earlier, WCF provides implicit data contracts for simple types such as strings and integers in order to exchange the data in platform neutral manner. It would violate the principles of SOA to simply return the Logical Device objects because other services might understand such an object.

As mentioned earlier, every WCF service must be hosted in some context. The chosen approach for this implementation is self-hosting. Creating the host is done in the Main() method by use of the ServiceHost class. The constructor of ServiceHost is provided with the service type, which in this case is of type ExchangeModel

```
ServiceHost hostExchangeModel = new ServiceHost
(typeof(ExchangeModel), baseAddressExchangeModel);
```

A platform neutral contract for the system must be created. The contract is used by the clients in order know which services to consume from the system. Besides, a platform neutral configuration file must be generated which includes information about the chosen binding for communication and which address to direct requests at. Configuring the system to use the binding WSDualHttp-Binding and to use IExchangeModel as contract for exposed services is handled with following

```
WSDualHttpBinding wsDualHttpBindingExchangeModel =
new WSDualHttpBinding();
hostExchangeModel.AddServicePoint(typeof(IExchangeModel),
 wsDualHttpBindingExchangeModel, "http://localhost:8002/");
```

Creation of the contract and configuration file is done with the tool svcutil by directing it at the baseAddressExchangeModel (after the service has been launched). Svcutil generates service model code, that is, a contract and a configuration file, from metadata about the service extracted from the baseAddressExchangeModel. In order to use svcutil, the system must first publish its metadata. This is achieved by use of the ServiceMetaDataBehavior class. By use of svcutil, the contract and configuration file both in plain text are generated. By importing these two files, the (WCF) client will know how to communicate with the system and what to consume from the system. The generated contract and configuration file are named ExchangeModel.cs and app.config, respectively and can be seen in the source code for the client in appendix B.2. Although not used, the WSDL file has also been generated by use of the tool disco. The WSDL file can be seen in appendix D. The WSDL file is platform neutral and can be applied by any platform that is capable of consuming web services. However the files generated by svcutil has been used in this thesis because both client and system are mapped on WCF.

IExchangeModel defines a callback contract of type IReportCallBackContract by use of following

```
[ServiceContract(CallbackContract =
typeof(IReportCallbackContract))]
```

This enforces all clients that want to communicate with the system to implement the methods defined in the callback contract. In the callback contract used for this thesis, the following methods are present

```
[OperationContract]
void OnCallBack(string reportMessage);

[OperationContract]
void SetCBEnabledStatusValue(bool enabled, string type);
```

Thus, the client must implement the two methods OnCallBack and SetCBEnabledStatusValues with signatures identical to the above. The method OnCallBack is used by the system, when reporting occurs. The system creates a report with a given message, and sends it to the client. The system will not be able to report to the client (invoke the OnCallBack method on the client) unless it knows how to contact the client. This is why the client must create a context around the callback contract and pass a reference to it in every call that it makes to the system. By using this reference, the system will be able to contact the client. Section 4.2 will describe in more details, how the client handles the callback contract. The client must keep its connection to the system established if it intends to receive spontaneous reports. Details regarding how to contact clients is the responsibility of the CallbackManager object, which will be described in section 4.1.1.

The second method of the callback contract is the SetCBEnabledStatusValue. The system uses this method to inform the client that a given control block has been activated or deactivated. It is used with reporting related control blocks, UBRCB and BRCB. The reason for its use is to be considered due to the MinRequestTime and MaxRequestTime within RCB. Outside this window, the system will deactivate the reporting. Because the status of reporting can be controlled by the system, the client must be informed, when changes of the state happen.

ExchangeModel

ExchangeModel must implement all the services as defined in IExchangeModel. This does not mean that ExchangeModel does the entire work itself. It delegates the tasks to the Information Experts which are the other objects. As mentioned in section 3, the ExchangeModel can be considered as a Controller (pattern). As an example, consider when the client wants to subscribe to a given data set for unbuffered reporting.

```
public void SetUBRCBValues(string dataSet, bool enable)
{
this.ubrcb.SetCBValues(dataSet, enable);
}
```

As can be seen in the code above, ExchangeModel delegates the task to the ubrcb object.

CallBackManager

The responsibility of CallBackManager is to keep track of the connections to the clients. When a client connects to the system, it first associates with the system. While association takes place, ExchangeModel retrieves the callback for the given client with following code

```
this.callback = OperationContext.Current.
GetCallbackChannel<IReportCallbackContract>();
```

Retrieval of the client callback is possible, because the client provides it every time it uses the system.

CallBackManager saves the callbacks for later use, namely for reporting. It uses objects of type Callback to store callbacks. Each Callback object is characterized by following three attributes

```
private IReportCallbackContract callback;
private int clientId;
private bool connectionEstablished;
```

The callback attribute is the actual address for the client, which the system uses to contact the client. The attribute clientId is used to identify, which client the callback belongs to. The attribute connectionEstablished is a Boolean indicating, if the connection to the client is established. When the client associates with the system, the connection state is set to true and it is assumed to be the case until the opposite is proven. The opposite is proven when reporting fails, and the Report object sets the connectionEstablished attribute to false for the particular client.

Note that the CallBackManager is a Singleton. It is natural to have a single central location, where this information is stored.

Associating

Association happens when the client connects to the system either for the first time or as a returning client due to earlier connections.

When association happens, ExchangeModel retrieves the client id, besides the callback for the client as described earlier. Retrieval of the client id is accomplished with following line of code

```
this.clientId = clientId;
```

Access control has not been a topic in this thesis, but the client id could be used for such purpose. The system uses the following code in order to model a simple access control

```
if(this.clientId != 0)
{
this.ReadyToContinue();
}
```

Only clients that have an id different from zero are authorized to use the system. This is an all or nothing approach, that is, either the client is granted access to the complete system, or it is not granted access at all. In reality the access

control would be finer grained than this. For instance by specifying which parts of the data model that the client must be able to see and use reporting and logging for. In case of access control based on client id, it must be considered, how the client id's are actually defined. It must be guaranteed that the id's are unique in order to make it possible to identify each client uniquely. It must also be considered that the client id must not be possible to be guessed in order to keep intruders out of the system. Alternatively, a password could be used together with the client id for authentication. Subjects related to improvement of the access control will be left open to be addressed in future work.

After successful association, the ReadyToContinue method makes sure that the client is allocated its reporting and logging resources, as well as resources for accessing the data model and the data set contents. Following code shows this

```
server = Server.Instance;
cbManager = CbManager.Instance;
this.lcb = cbManager.GetLCB(clientId);
this.ubrcb = cbManager.GetUBRCB(clientId);
this.brcb = cbManager.GetBRCB(clientId);
this.dataSetManager = new DataSetManager();
```

Note that the control blocks are delivered to the client according to the client id. This ensures that the client can retrieve its already existing control blocks, if it is reconnecting rather than connecting for the first time. As can be seen in the code above, CbManager is responsible for delivering the control blocks, making it possible for the client to use them. Also note that the Server and CbManager objects are Singletons. Same instance of these objects is used by all objects.

DataSetManager

DataSetManager is responsible for creating and keeping track of the data set contents. It has two lists of data sets, that is, one for reporting related data sets and one for logging related data sets as can be seen below

```
private List<DataSet> dataSetsLogging;
private List<DataSet> dataSetsReporting;
```

DataSetManager needs to know the data model, represented by the Server object, in order to create the data sets.

As mentioned earlier, all the data set rules are preconfigured. The data sets are defined with names in accordance with IEC 61400-25. The code below shows some data sets

```
private DataSet turCmLog;
private DataSet turStLog;
private DataSet hiUrgAlm;
```

DataSetManager creates the data sets, by specifying what to look for in the data model as specified in IEC 61400-25. The code below illustrates this

```
this.turCmLog = this.SetDataSetValues(this.turCmLog, CDC.CMD,
"ActSt", FunctionalConstraint.CO)
this.turCmLog = this.SetDataSetValues(this.turCmLog, CDC.CMD,
"ActSt", FunctionalConstraint.ST)
```

The SetDataSetValues method will iterate through the data model looking for mathces. Every time a match is found, the data attribute reference is added to the data set.

As mentioned earlier there is two types of signatures for methods creating data sets. One signature is the parameters (CDC, dataGroup, functionalConstraing). The above code used this signature. The other signature consists of the parameters (CDC, data attribute name). The code below shows the creation of a data set using this signature

```
this.turTmLog = this.createDataSetValuesCDCAtt(this.turTmLog,
CDC.TMS, "tmTot");
```

Note that names of the functional constraints and CDC's are defined in their own classes in order to avoid typing in strings manually. Both classes are static, thus instantiation is not necessary.

After the data set has been created, it is added to the list of either reporting or logging related data sets. The TurCmLog data set is logging related and is added to the logging related data sets

```
this.dataSetsLogging.Add(this.turCmLog);
```

The process is similar for reporting related data sets.

By keeping track of which data sets that belong to reporting and logging, it is possible to provide the RCB's with reporting related data sets and the LCB's with logging related data sets, when they ask for it. The method GetDataSetsReporting returns a list of reporting related data sets and the method GetDataSetsLogging returns a list of logging related data sets.

DataSetManager is also responsible for keeping track of data set contents. For reporting and logging, it must be determined if a given data attribute is referenced by a given data set. DataSetManager answers this question by the AttributeInDataSet method

```
public bool AttributeInDataSet(DataSet dataSet,
string dataAttributeReference)
{
if(dataSet.AttributeInDataSet(dataAttributeReference))
{
return true;
}
else
{
return false;
}
}
```

Server

The system has a single Server object, which is a Singleton. The Singleton is achieved by following approach

```
private static readonly Server instance = new Server();
```

The readonly modifier ensures that the Server is created only once. Future attempts to access the server will use the public get method in the Server object

```
public static Server Instance
{
get{ return this.instance; }
}
```

The Server object is created even before any clients have contacted the system. It is achieved in the Main() method

```
Server server = Server.Instance;
```

Considering that the Server uses system resources, the creation of it could be delayed until the first client connects to the system. However the Server is a central element of the system, therefore it is being created early. It is simple to delay the creation of the Server by simply out commenting the above code in the Main() method, in which case the Server will be created when the first client connects to the system.

The Server holds objects of type LogicalDevice

```
private List<LogicalDevice> logicalDevices;
```

The LogicalDevice object in turn holds object of type LogicalNode, which holds objects of type DataEntity, which holds objects of type DataGroup. The DataGroup holds the basic building blocks, that is, objects of type DataAttribute. Creation of the DataAttribute objects is the responsibility of the DataAttributeManager. The DataAttributeManager object is created at the first creation of the Server object. Thus, the DataAttributes are ready to use when the Server is going to initialize its data model.

After the Server has been created it initializes its data model contents by calling the method

```
Initialize()
```

This method reads the WPPCL file and creates the data model contents according to this. For instance, when a logical device is detected in the WPPCL file, a new logical device is created and added to the server by the AddLD method

```
private void AddLD(logicalDevice ldName)
{
this.logicaldevices.Add(ldName);
}
```

The Server object uses the built in class XmlReader class to provide fast reading from the WPPCL file. XmlReader is forward-only, but that is sufficient for reading the WPPCL file.

When enabled data attributes in the WPPCL are detected, they are added to relevant location in the data model by use of following method

```
private void CreateNewDataAttribute
(string name, int ld, int ln, int de, int dg, int da, bool isEnabled)
{
if(isEnabled.ToLower() == "true")
{
DataAttribute dataAttribute = new DataAttribute();
dataAttribute = this.dataAttributeManager.GetDataAttribute(name);
this.LogicalDevices[ld].LogicalNodes[ln].DataEntities[de].
DataGroups[dg].AddDataAttribute(dataAttribute);
}
}
```

The data attribute is created as an empty data attribute, and its value is retrieved from the DataAttributeManager according to its name.

After initialization of the data model the Server contains the data model that reflects the contents of the WPPCL file. But the data attributes within the data model does not yet hold data from a wind power plant. This is achieved by calling the UpdateServer method, which gets data from the WPPDataGenerator. What the UpdateServer methods does is that it runs through all the data attributes of the Server and for each of them gets a value from the WPP data generator. The UpdateServer method is capable of recording the value for at data attribute before and after it was updated. For every data update the UpdateServer informs the EventMonitor object

```
this.eventMonitor.CheckAttribute(dataAttributeReference,
objValue, objNewValue, triggerOption);
```

The parameters objValue, objNewValue and triggerOption are sufficient for determining, if condition for the trigger option has been met. The parameters objValue and objNewValue are the values for the data attribute before and after the update, respectively. The triggerOption is the trigger option associated with the data attribute. The parameter dataAttributeReference is used for logging and reporting in order to know, which data attribute caused the event.

WPP Data Generator

WPPDataGenerator is a simple object, generating random data. Seen from the perspective of the system, it is a wind power plant that generates data. The object is not considered as a realistic WPP data generator. Actually it is just a random data generator. It is a candidate for future work to improve this part.

EventMonitor

After being contacted by the Server, EventMonitor has to determine if the trigger condition has been met. This is determined in the CheckAttribute method

```
if(oldValue != newValue && triggerOption.ToLower() ==
Trigger.dhcg.ToLower())
```

The reason for only checking for the dchg trigger is that no data attribute in IEC 61400-25 is associated with the dupd trigger. When it comes to the trigger qchg, it is treated the same way as dchg as mentioned earlier. However, it would be

straightforward to modify the EventMonitor class to be able to handle dupd and qchg. Dupd will always be true, since the Server contacted the EventMonitor in the first place. Qchg would be determined by comparing old and new value for the quality field in the data attribute.

If it is determined that a trigger condition has been met, EventMonitor must inform the CBMediator. This is achieved by

```
this.cbMediator.Notify(dataAttributeReference, oldValue,
newValue, triggerOption)
```

CBMediator

CBMediator has indirect knowledge about all control blocks in the system because it knows the CBManager. What the CBMediator does is that it informs every control block when informed by the EventMonitor. Then it is up to each control block determine if reporting and logging must occur. Why not just let EventMonitor inform all control blocks rather than having the CBMediator to do it? It is in order to obtain high cohesion, that is, objects do not fulfill unrelated tasks.

CB

Determining if reporting and logging must occur depends on the following criteria

- Does the data attribute belong to any data set that the control block has subscription for?

If this is the case then reporting and logging happens. If it is not the case, no reporting or logging occurs. The following method in the CB object investigates, if conditions for reporting or logging are satisfied

```
protected bool ConditionsForAction(string dataAttributeReference)
{
foreach(DataSet dataSet in this.subscriptions.Subscriptions)
{
if(this.dataSetManager.AttributeInDataSet(dataSet,
dataAttributeReference))
{
return true;
}
}
return false;
}
```

The method runs through all the subscribed data sets and for each one it checks if it references the data attribute. If it is the case, then reporting or logging must occur. If not, then reporting or logging does not occur.

The reason that the name of the method is ConditionsForAction is that it relates to both reporting and logging. That also explains why the method is protected. It is because the method is in the abstract class CB. The control blocks all inherit from CB and by making the method protected, all the inherited classes are able to use the method.

Reporting is described in section 4.1.2 and logging is described in section 4.1.3.

CBManager

CBManager is the Information Expert regarding knowledge about the control blocks in the system. It has a list for each type of control block

```
private List<LCB> lcb;
private List<UBRCB> ubrcb;
private List<BRCB> brcb;
```

When a client connects to the system via the ExchangeModel object, control blocks must be made ready to use. If the client is a first time visitor, new instances of control blocks must be created and associated with the client. The code below illustrates that the client is a first time visitor, which is why is gets a new LCB.

```
LCB newLCB = new LCB(clientId);
lcb.Add(newLCB);
return newLCB;
```

The control block is returned to the ExchangeModel, where the client will be able to use it.

The reason that CBManger must keep track of which control block belongs to which client is in order to be able to handle reconnecting clients. That is, if the client after its first visit has disconnected from the system and later returns to the system again, then it shall use its already existing control blocks rather than getting new instances. Determining if a client already has an LCB is achieved by following

```
if(lcb.Exists(delegate (LCB lcbReturn){return
lcbReturn.ID == clientId;)})
```

If it is a returning client, then its LCB must be in the list of LCB's in the CBManager. It must be found in the list and returned. This is achieved by following code

```
return lcb.Find(delegate(LCB lcbReturn){return
lcbReturn.ID == clientId;});
```

The process is similar for UBRCB and BRCB.

DataAttributeManager

This object creates all the data attributes that [61400-25-2] has defined (no knowledge of data attributes from [61850-7-2]). It creates objects of type DataAttribute. Each DataAttribute has a

```
private string name;
private string type;
private string functionalConstraint;
private string triggerOption;
private string description;
```

The Server object uses DataAttributeManager when creating the contents of the data model by requesting a data attribute with a given name. If DataAttributeManager does not have a data attribute with that name, an "IEC 61850 attribute" is returned. This approach is used because [61400-25-2] inherits data attributes from [61850-7-2], but DataAttributeManager does not know about these. This is the case because there is a quite large amount of data attributes, and it takes a lot of time just to type them into the DataAttributeManager.

At the moment, most data attributes have the same type, functionalConstraint, triggerOption and description. Again, this has been chosen because other aspects of the system have been given higher priority when doing the implementation, rather than using a lot of time on the typing. It does not change any behavior of the system that the data attributes do not have their real data typed in. For instance reporting and logging is unaffected of this. Before deployment of the system happens in the future, typing in all details as defined in [61400-25-2] must take place as well as data attributes inherited from [61850-7-2].

4.1.2 Reporting

UBRCB and BRCB inherit from the abstract class RCB (which in turn inherits the abstract class CB). The RCB includes all the reporting related tasks, such as sending the report to the client and keeping track of the activation period for the reporting (specified by minRequestTime and maxRequestTime). By inheriting from RCB, which inherits from CB, the reporting related objects UBRCB and BRCB have all they need in order to take responsibility for making the reporting happen.

RCB defines an abstract method named ReportingFailed, which the inheriting classes (UBRCB and BRCB) must implement. RCB invokes this method when reporting fails.

RCB uses the Report object to do the actual reporting. It tries to send the report, and if it fails, it updates the state of the connection for the client to false and throws an exception, to inform the RCB that reporting went wrong

```
this.callbackManager.SetConnectionState(clientId, false);
throw new Exception();
```

This exception is caught by the RCB, which then invokes the method ReportingFailed, which by use of polymorphism makes sure that the ReportingFailed method in the relevant class (UBRCB or BRCB) is executed. UBRCB does nothing in case of failed reports while BRCB buffers them.

4.1.3 Logging

LCB inherits from the abstract class CB. When logging must occur, log entries are inserted into the log. LCB has access to the persistent storage by use of a DbHandler object

```
private DbHandler dbHandler;
```

The persistent storage is realized by use of an Access database. A single table in the database is used for the logging. Since this thesis serves as proof of

concept, Access has been considered sufficient. Use of MS SQL or MySQL must be considered when deploying the system, in order to achieve better scalability regarding the number of clients and performance in general.

The DbHandler is initialized with the clientId

```
this.dbHandler = new DbHandler(clientId);
```

The motivation for passing on the clientId to DbHandler is that every LCB object inserts entries into the same log. The clientId can be used as part of the log entries in order to make it possible to distinguish between log entries belonging to different clients. At the moment, DbHandler does not make use of the clientId. It will be left to future work to add a field named clientId to the table in the database. At the moment all clients can retrieve the complete part of the log.

4.2 Client

The client consumes services exposed by the system. First, the client must import the contract and configuration generated from the metadata of the system in order to know the services exposed by the system and how to communicate with the system. The client offers a graphical user interface as presented in section 3.4. Before the client is able to use to the system, it must associate with it in order to be identified by the system. Source code for the client including the contract (ExchangeModel.cs) and configuration file (app.config) can be found in B.2

4.2.1 The callback

Before the client is able to communicate with the system, it must implement the callback as specified by the contract and create a context around it. The context must be passed on to the system. This is achieved by

```
IExchangeModelCallback callback = new MyCallBack(this);
InstanceContext context = new InstanceContext(callback);
this.client = new ExchangeModelClient(context);
```

As mentioned before, passing the context to the system is necessary, because the system must now how to get to the client when reporting must occur.

The class MyCallBack implements the callback contract defined by IExchangeModelCallback. The method OnCallBack looks like this

```
public void OnCallBack(string attribute)
{
form31.ReportingHappened(attribute);
}
```

Form31 is an alias for the graphical user interface. In other words, every time the system reports to the client, the OnCallBack method is invoked. In turn, the OnCallBack method informs the graphical user interface, so it can write the report to the screen. This is how the publisher/subscriber pattern is experienced from the client side.

The method SetCBValues is also implemented by the client, as specified in the callback contract. This method is used by the system, when reporting related control blocks are activated or deactivated. By doing so, the client always knows the state of the reporting related control blocks.

4.2.2 Association

The client uses an integer representing its client id to associate with the system. The service Associate is used in the following manner

```
this.client.Associate(13);
```

4.2.3 RetrieveDataModelContents

If association was good, then the client is able to use the system. The client starts with retrieval of the data model contents by use of the method PopulateTree(). This method uses the services related to the use case RetrieveDataModelContents [2].

After retrieval of data model contents, the client retrieves data sets, both reporting related and logging related. This is achieved by the method PopulateListBoxDataSets() which makes use of the services GetDataSetsReporting and GetDataSetsLogging. Note that these two services are not a part of the information exchange model according to [61400-25-3]. However, they are used as services in this thesis, because the client needs to retrieve the data sets before it can use reporting and logging.

4.2.4 GetDataSetValues

Retrieval of data attributes referenced by a given data set is possible by selecting a data set and pressing the button "View". This will invoke the service GetDataSetValues.

4.2.5 Subscriptions

The client checks if it has already subscribed to any data sets for reporting and logging with the method GetSubscribedDataSets(). This is possible because the system stores the reporting and logging related resources allocated for the client when the client disconnects. If this was not the case, reporting and logging would not have been possible when the client disconnected. The client gets its subscriptions in the same state that it left them when it disconnected from the system.

The client is able to manage its subscriptions for reporting and logging by use of the buttons "Subscribe" and "Unsubscribe" associated with each control block.

4.2.6 Buffered reports

The client checks if it has any buffered reports in the system by use of

[2] GetServerDirectory, GetLogicalDeviceDirectory, GetLogicalNodeDirectory, GetDataEntityDirectory, GetDataDirectory

```
this.client.CheckForBufferedReports();
```

If buffered reports exist, the client retrieves them by use of the method GetBufferedReports. The reports are shown in the graphical user interface in the same manner as reports in general, namely by use of the method ReportingHappened.

4.2.7 Reporting

As mentioned before, when reporting occurs, the OnCallBack method invokes the method ReportingHappened, in order to output the report to the graphical user interface.

4.2.8 Logging

As opposed to reporting, log entries are not received spontaneously. The client must specify which parts of the log shall be retrieved by specifying filter conditions in terms of (time1, time2) or (time, entryId). The log is then retrieved by pressing the button ”Query log” and the contents will be shown.

4.3 WPPCL file

The file has been created and can be found in appendix C. Below, a small example from the file is presented.

```
<server servername = ”61400-25”>
  <ld ldname=”logicalDevice1”>
    <ln lnName=”wtur”>
      <data dataName=”AvlTmRs” commonDataClass=”TMS”>
        <dataGroup dataGroupName=”manRs”>
          <dataAttribute dataAttributeName=”ctlVal” enabled=”false” />
        </dataGroup>
        <dataGroup dataGroupName=”hisRs”>
          <dataAttribute dataAttributeName=”ctlVal” enabled=”true” />
```

The structure and contents of the created WPPCL file is in accordance with the requirements. Note the field named ”enabled”. This is the field that WPPCL Editor configures.

The server element could be omitted because the WPPCL file should not be intended for a specific system. However by understanding the server element as an option for writing details about the WPPCL file, its use could be justified. For instance it could specify the version for the WPPCL file and the company that created it or similar.

4.4 WPPCL Editor

Similar to the data model in the system, WPPCL Editor contains a data model. While the hierarchy is identical for the two data models, there can be a difference in the number of included data attributes. WPPCL Editor includes all

data attributes, also the disabled ones. This is not the case with the data attributes in the system where only enabled data attributes are added to the data model. With WPPCL Editor it is necessary to add both enabled and disabled data attributes to the data model, because it shall be possible to configure the WPPCL file.

When WPPCL Editor is launched, it reads the WPPCL file by use of the built in .NET XMLReader class. This class provides fast, non-cached, forward-only access to the xml file, this is just what is needed.

After the WPPCL file has been read, the data model contents is visualized in the tree structure as shown in the graphical user interface of the WPPCL Editor (described in section 3.3). Enabled data attributes have the color green, and disabled data attributes have the color red. By selecting a data attribute it is possible to enable or disable it. The data model is updated every time a data attribute is enabled or disabled, and the relevant part of the tree is repopulated.

Note that only the relevant part of the tree is repopulated rather than repopulating the complete tree. Repopulating the complete tree is unnecessary and would take more time. Only populating the relevant part of the tree means to only repopulate the leaf of the tree, where the data attribute is present. The leaf is always at the data group level, because they group together data attributes. The method PopulateLeaf is used to populate the leaf, where the data attribute is present. Actually, the PopulateLeaf method is also used by the initial routine while populating the complete tree. The PopulateLeaf method is then invoked for each leaf of the tree.

When the editing has finished, it is possible to save the changes by pressing the ”Save” button. This overwrites the old WPPCL file by use of the method SaveWPPCL(). This method uses the XMLWriter class, which like XMLReader, provides a forward, non-cached, forward-only approach for generating xml.

Source code for WPPCL Editor can be seen in B.3.

4.5 Conclusion

This chapter has mapped the IEC 61400-25 compliant system to the web service mapping as proposed in [61400-25-4].

The system initializes contents of its data model by use of the WPPCL file. The WPPCL file has been created, and it reflects the data model contents by a wind power plant. The file would have been created by the wind power plant manufacturer, but since no such wind power plant has been used in this thesis, it has been the task of this thesis to create the file.

The reporting mechanism makes use of the publisher/subscriber pattern. That is, reports are sent spontaneously to the client. In order for the system to be able to contact the client with web services, the WSDualHttpBinding in WCF is used.

The system specifies a contract of type ExchangeModel, which is a representation of the Information Exchange Model. The system exposes metadata about communication and services, which svcutil uses for creating the ”abc” for the system, that is, address, binding and contract. The client uses the ”abc” in order to know how to contact the system and what to consume from the system. Although generated, the WSDL file is not used directly by the client. This is because svcutil generates the service model code for a WCF client directly. Be-

hind the scenes however svcutil generates the service model code by using the WSDL exposed by the system in terms of metadata. The WSDL is platform neutral and can be used to create a client on any platform that is capable of consuming web services to create a client.

The IExchangeModel defines a callback contract of type IReportCallBack-Contract. This callback contract must be implemented by the client and a context must be created around it. Every time the client contacts the system, this context must be provided. This ensures that the system will be able to report to the client spontaneously. Naturally, the client must be running and online in order for reports to be delivered spontaneously.

The system stores subscriptions for reporting and logging in memory, rather than in a persistent manner. This is also the case with contact details about the clients, represented by the CallBackManager object. In a proof of concept model, this approach is considered sufficient, but the consequence is that all subscriptions and contact details for clients (CallBackManager) will be lost due to a system shutdown (or crash for that matter). An approach for avoiding this would be to use persistent storage. At startup, the system will check the persistent storage for subscriptions and client connections, and create control blocks according to this. Naturally, while the system is being restarted, neither reporting nor logging will take place. For the reporting, it has to be activated by the client, even if it was active at the time, the system was shut down. Logging is always active, and it will continue logging according to the subscriptions.

The mapping of the system has not followed the mapping definition exactly, but rather used it as guideline. For instance the Server object in the system shall be named tServer according to [61400-25-4].

The system is prepared to a multi-client environment, where each client is identified and delivered its own resources for reporting and logging and use of the system in general. However, further steps must be taken in order to make it safe to use the system in a multi-client environment. For instance, the Server object is a Singleton, which is created one time and only one time at the lifetime of the system. But the creation of the Server object is not thread safe at the moment. This must be ensured, to avoid having multiple clients creating multiple Server objects due to simultaneous access. The same holds for the other Singletons in the system, namely CBManager and CallbackManager.

Another challenge for use in a multi client environment is related to the logging. Since all the logging is done to the same persistent storage, there must be a field in the log entries that specifies which log entries belong to which client. Then implicit filtering on this condition would be done every time clients would query the log and they would only retrieve their own part of the log. This modification would be straightforward to do, because the DbHandler already knows the id of the client it serves. It must be noted that introduction of the new field must be a supplement to the unique id of each log entry rather than a replacement. Each log entry must still have a global unique id.

The system is capable of balancing its load if too much reporting happens. This is achieved by altering the size of the window determined by MinRequest-Time and MaxRequestTime. At the moment, the size of the window is static. MinRequestTime has been set to zero, that is, reporting is activated immediately when the client activates it. In order to have dynamic control of the window, system resources must be monitored, and the size of window adjusted accordingly.

A client that is capable of consuming the services exposed by the system has been implemented. The client provides a graphical user interface, representing the SCADA, which makes it possible to configure and consume the reporting and logging, besides general use of the system.

When clients reconnect to the system, they are delivered their already existing control blocks with their existing subscriptions, which was created at the first connect.

In general, when clients connect to the system, they ask if there are any buffered reports waiting for them. If it is the case, the client will ask for the reports, and the system will deliver them. This is a request-response approach, where the normal reporting mechanism uses the publisher/subscriber approach. At the moment, if buffered reports exist, the client asks for all of them in one request and the system delivers them. For this thesis, this approach has done the job. However, the mapping environment for the system and client has been on a desktop computer with relative large memory capabilities (512MB). If the mapping environment had less resources, for instance on handheld devices, this could turn out to be an issue. Similar to the system, which can balance its load due to reporting by use of MinRequestTime and MaxRequestTime, the client must be able to balance its load when retrieving buffered reports. This can be achieved by letting the client retrieve the buffered reports in groups, which has a size determined by the client, rather than retrieving all of them in one time. This could be a subject, if the mapping environment was realized on relative resource weak devices.

A WPPCL file has been created in xml. It reflects the data model structure as defined in [61400-25-2]. Each data attribute has a field named "enabled" which indicates if the data attribute is enabled or disabled. WPPCL Editor manipulates this field when configuring the WPPCL file.

WPPCL Editor with a graphical user interface has been implemented. The WPPCL Editor shows the data model represented by the WPPCL file in a tree structure (like the tree in the SCADA). It is possible to edit the data model by clicking at data attributes and enabling or disabling them. Coloring has been used to improve the usability for the WPPCL Editor. Green indicates that data attributes are enabled, and red indicates that data attributes are disabled.

To improve performance only the leaf where the selected data attribute is present, is repopulated, rather than repopulating the complete tree for every editing. When the editing is completed, the WPPCL overwrites the old WPPCL file. When the system initializes next time, the modification of the WPPCL file will have effect.

WPPCL Editor uses the .NET built in classes XMLReader and XMLWriter to read and write the WPPCL file, respectively. These classes provide fast performance for reading and writing xml.

An approach for improving WPPCL Editor would be to make it possible to enable and disable contents of the data model at a higher level than the data attribute level. This must be considered because there is a lot of data attributes in the typical data model, and it would take a lot of time to enable or disable large groups of data attributes by doing it separately for each data attribute. This modification could be accomplished by making it possible to edit the data model at the level of data groups, data entities, logical nodes or even the logical device level. An algorithm would be necessary for traveling all contents of the selected level, and doing the edit as specified by the user. For instance, if the

user has chosen to enable a given logical node, all data attributes within the data entities and data groups for the selected logical node must be enabled.

Chapter 5

Test

This chapter will test the implemented system, client and WPPCL Editor. Black box testing will be used. Section 5.1 tests the external and internal behavior of the IEC 61400-25 compliant system by use of the client. In the process, the client is also tested. The test includes the reporting and logging and general use of the system. Both unbuffered and buffered reporting will be tested. Section 5.2 tests the WPPCL Editor. Section 5.3 concludes the chapter.

5.1 IEC 61400-25 compliant system

The system has been tested by use of the client, thus the client has also been tested. Only testing with a single client has been carried out. Although the business logic in the system is ready to handle multiple clients, issues such as concurrency has not been dealt with. In general, focus has been on the proof of concept model rather than being able to handle multiple clients simultaneously.

5.1.1 Association

Test name: Association.
Execution: Client connects to the system.
Expected outcome: Client must be identified and either granted or denied access.
Actual outcome: Client is identified and granted or denied access.

5.1.2 RetrieveDataModelContents

Test name: GetServerDirectory.
Execution: Client uses the service GetServerDirectory.
Expected outcome: Client must retrieve logical devices in the system.
Actual outcome: As expected.

Test name: GetLogicalDeviceDirectory
Execution: Client uses the service GetLogicalDeviceDirectory and specifies a logical device.
Expected outcome: Client must retrieve logical nodes in the specified logical device.

Actual outcome: As expected.

Test name: GetLogicalNodeDirectory
Execution: Client uses the service GetLogicalNodeDirectory and provides it with (logical device, logial node).
Expected outcome: Client must retrieve data entities in the specified (logical device, logical node).
Actual outcome: As expected.

Test name: GetDataEntityDirectory
Execution: Client uses the service GetDataEntityDirectory and provides it with (logical device, logical node, data entity).
Expected outcome: Client must retrieve data groups in the specified (logical device, logical node, data entity).
Actual outcome: As expected.

Test name: GetDataDirectory
Execution: Client uses the service GetDataDirectory and provides it with (logical device, logical node, data entity, data group).
Expected outcome: Client must retrieve data attributes in the specified (logical device, logical node, data entity, data group).
Actual outcome: As expected.

5.1.3 GetDataSetValues

Test name: GetDataSetValues
Execution: Client uses the service GetDataSetValues and provides it with the name of the data set.
Expected outcome: Client must retrieve the data attributes referenced by the given data set.
Actual outcome: As expected.

5.1.4 SetSubscription

Test name: SetSubscription
Execution: Client uses the service Set[ControlBlock]Values for one of the control blocks, for instance UBRCB.
Expected outcome: Client must be able to set its subscriptions, that is, add or remove subscription for the given control block.
Actual outcome: As expected.

5.1.5 GetSubscriptions

Test name: GetSubscriptions
Execution: Client uses the service Get[ControlBlock]Values for one of the control blocks, for instance UBRCB.
Expected outcome: Client must retrieve a list of subscriptions for the given control block.
Actual outcome: As expected.

5.1.6 Reporting

Unbuffered reporting

Test name: Unbuffered reporting
Execution: Client subscribes to reporting related data sets and waits for the system to publish reports.
Expected outcome: If the client is connected to the system, it must receive reports spontaneously. The received reports must be shown in the SCADA. On the other hand, if the client is not connected, the reports must be discarded.
Actual outcome: As expected. It must be noted, that the system tries to send a report for 60 seconds, before it gives up and throws a TimeOutException. This exception causes the report to be discarded, and following attempts for reporting will be discarded rather than trying to be sent. This is indeed the intention, so it is good. However, if the updating mechanism in the server uses an interval of length 60 seconds or less for the updating, unexpected behavior happens in case of not established connection. The explanation to this is that while existing reports are trying to be sent, the system starts a new round of updating with more potential reports. The 60 seconds before the TimeOutException is thrown has not elapsed yet, thus sending the new reports will be attempted. Depending on the interval for updating, a couple of minute may elapse, before all the reports that were tried to be sent are discarded. In order to avoid this behavior the system shall not do the updating with intervals of 60 seconds or less. Another approach would be to use a timer, that throws an exception before the 60 seconds has elapsed. Then it would be possible to do the updating with shorter intervals than 60 seconds. The important part is that the exception is thrown before a new updating round is initiated.

Buffered reporting

Test name: Buffered reporting
Execution: Client subscribes to reporting related data sets and waits for the system to publish reports.
Expected outcome: If the client is connected to the system, it must receive reports spontaneously. If the client is disconnected, the system must buffer the reports. At next connection with the system, the client must retrieve its buffered reports by request-response. The reports must have a unique id each. The received reports must be shown in the SCADA.
Actual outcome: As expected. Regarding the length of the interval for doing the updating, the same applies to buffered reporting as for unbuffered reporting. If too short interval for updating is used when the connection is not established, the reports are not inserted into the buffer chronologically due to the fact, that reports from different rounds of updating are involved. This can be seen by the client, when it reconnects to the system and retrieves its buffered reports. The buffer is a fifo buffer, so it is not causing the unordered reports. Investigating the unbuffered reporting, it would also be the case, that they are not discarded chronologically. However, it is not of importance, in which order reports are discarded. As mentioned for unbuffered reporting, having a custom timer that throws an exception before next updating round is initiated can be used to avoid trying sending reports when the connection is not established.

5.1.7 Logging

Test name: QueryLogByTime
Execution: Client uses the service QueryLogByTime and specifies two times.
Expected outcome: The log entries logged within the two times specified must be returned to the client.
Actual outcome: As expected. However, if the query returns a large number of log entries, WCF will throw an exception due to the limit of the WSDualHttpBinding. This makes good sense, since it can not be of interest to send too much data in a response to a request in the web service environment. The connection speed between client and system could be a low bandwidth connection in which case it would be better to send the log entries in smaller groups. It must be the task of the system to find an appropriate size for returning the log entries.

Test name: QueryLogAfter
Execution: Client uses the service QueryLogAfter and specifies a time and a log entry id.
Expected outcome: The log entries logged after the specified time and with an entry id larger than the supplied id shall be returned to the client.
Actual outcome: As expected. In case of large number of log entries that the query returns, WCF will throw an exception, as it happens with QueryLogByTime.

5.1.8 Updating mechanism

Test name: Update values for data model contents
Execution: At regular intervals, the system updates the values for each data attribute in its data model. This is achieved by polling the wpp data generator for random data.
Expected outcome: Each data attribute in the data model in the system must be delivered a value from the wpp data generator.
Actual outcome: As expected.

5.2 WPPCL Editor

Testing of WPPCL Editor is focused on the editing capability of the WPPCL file. When launched, WPPCL Editor reads the WPPCL file according to a path written in the business logic and shows the contents in a tree structure. Enabled data attributes have the color green and disabled data attributes have the color red. By clicking a data attribute and selecting either to enable it or disable it, WPPCL Editor saves the change in memory and updates the leaf of the tree, where the data attribute is located. Trying to enable a data attribute that is already enabled is possible, although an alternative approach would be to inform the user, that the data attribute is already enabled. The same holds for disabled data attributes. WPPCL Editor saves the changes to the file specified in the business logic, when the user hits the ”save” button. If the editing shall not be saved, WPPCL Editor can simply be closed.

It must be noted that WPPCL Editor saves the changes in memory until the ”save” button is explicitly used. This approach provides fast perfomance, but if the computer crashes before saving the changes, all changes will be lost.

However this will not have any effect on the WPPCL file. What happens if the computer crashes while writing the WPPCL file? WPPCL Editor writes a new WPPCL file each time the "save" button is pressed rather than editing the old file. Although not investigated, most likely the WPPCL file would end suddenly where the computer crash took place. This can be used as a argument for creating a new file with WPPCL Editor rather than overwriting the old one. If the computer crashes while creating the new WPPCL file, the old WPPCL file will still be intact.

5.3 Conclusion

This chapter has tested the implemented IEC 61400-20 system by use of the client. In this process, the client has also been tested. Furthermore, WPPCL Editor has been tested.

The IEC 61400-25 compliant meets its requirements. A topic that must be considered, is the length of the interval for updating the values for the data model contents in the system. At the moment, the system waits for the underlying transport mechanism in WCF to throw an exception, if an error was experienced while trying to send the report. In such case, a TimeOutException is thrown. 60 seconds is the time that elapses, before the exception is thrown. This causes troubles for the system, if it updates values for the data model contents using intervals of 60 seconds or less, because before it is detected that the connection is not established, new reports are being tried to be sent. Two solutions for this challenge can be used. Either the length of the interval for updating must be more than 60 seconds, or a custom timer must be used in order to throw an exception before the next updating starts.

WPPCL Editor meets its requirements and passes the test. It is simple to use, and no errors in the WPPCL file can be caused by use of WPPCL Editor if the computer does not crash while writing the WPPCL file. However if it crashes, the old WPPCL file will be intact in case WPPCLEditor is configured (in business logic, not by user) to write a new WPPCL file rather than overwriting the old one.

Removing the error prone aspect of manual editing was the main motivation for creating WPPCL Editor and it can be concluded that it passes the test.

An approach for improving WPPCL Editor would be to let the client specify the path to the WPPCL file, and when saving the changes, it shall be possible to specify, if the old WPPCL file must be overwritten, or if a new WPPCL file must be created. This would allow creating different configuration profiles for the contents of the data model. However, this is also possible at the moment, but it involves manual work by the user. Before changes to the WPPCL file is saved by WPPCL Editor, the contents of the current WPPCL file is copied and saved into another file. When saving the changes by WPPCL Editor, two WPPCL files will be a reality. Before launching the IEC 61400-25 compliant system, the relevant version of the WPPCL must be put in the location where the system reads it from.

Chapter 6

Conclusion

This chapter concludes the thesis. Section 6.1 presents the results of the thesis. Section 6.2 presents the contributions of the thesis. Section 6.3 discusses the results and proposes subjects for future work.

6.1 Results

This thesis has created a proof of concept system in compliance with IEC 61400-25 with focus on reporting and logging. The system supports both unbuffered and buffered reporting. The reporting mechanism, in general, makes use of the publisher/subscriber pattern. Spontaneous reports are sent from the system to the client, when dictated so by the internal updating mechanism in the system and the state of subscriptions. The client retrieves its buffered reports when reconnecting to the system by use of request-response approach. Each client has its own set of control blocks in order to configure its own reporting and logging. The system makes use of the time window delimited by MinRequestTime and MaxRequestTime in order to balance load caused by reporting. Logging is done to persistent storage. Log entries can be queried at a later time.

Besides the system, a client has been created in order to consume the services provided by the system. A graphical user interface has been created for the client, representing a SCADA. The client is able to configure and consume reporting and logging besides general use of the system. The SCADA is used by humans to interact with the client. The contents of the data model in the system are visualized by use of a tree in the SCADA.

Out of scope topics for IEC 61400-25 have also been addressed in order to create a working system, which is possible to test. A WPPCL file, which specifies the contents of the data model for a given (imaginary) wind power plant, has been created. The WPPCL file is used by the system to initialize the contents of its data model in order to reflect the contents of the data model provided by a given wind power plant. Using the WPPCL file makes the system capable of representing all IEC 61400-25 compatible wind power plants.

Out of topic for IEC 61400-25 is also the WPPCL Editor, which has been created. It makes it possible to edit the WPPCL file. WPPCL Editor provides a graphical user interface making it intuitive to edit the WPPCL file avoiding the error prone aspect of manual editing with a text editor.

Another out of topic subject for IEC 61400-25 has been the WPP data generator, which has been created as a random data generator. However the system treats its data like if it comes from a real wind power plant.

Below the results will be described in more details.

6.1.1 IEC 61400-25 compliant system

Information model

The IEC 61400-25 compliant monitoring system has an information model in accordance with [61400-25-2]. This includes the hierarchical data model (from server element down to data attribute element), it includes the control blocks for reporting and logging and it includes the data sets. Only data attributes from [61400-25-2] has been included in the system. This means that inherited data attributes from [61850-7-2] has been left out due to large amount of typing work. The typed in data attributes from [61400-25-2] have their correct names, but does not reflect other values such as ”description” or ”functional constraint” correctly. Rather, most data attributes have the same values. Other parts of the thesis have been given higher priority than typing in a lot of data that does not change the operation of the system. The behavior of reporting and logging, and the system in general, is unaffected by the fact that most data attributes have the same values.

Data attributes are referenced by a unique path in the format

```
LogicalDevice.LogicalNode.DataEntity.DataGroup.DataAttribute
```

The term DataGroup has been proposed by this thesis as an abstraction level between data entities and data attributes. It is necessary in order to address each data attribute by a unique path.

The system uses preconfigured data sets. At startup, after the contents of the data model has been initialized, the system searches the data model for data attributes, which match the rules for the data sets as specified in [61400-25-2]. Each time a match is found, the data set adds a reference to the data attribute in the form of a unique path as shown above.

Information exchange model

The system exposes the contents of its information model in accordance with the abstract service definitions in [61400-25-3]. Most, but not all, services defined by [61400-25-3] is exposed by the system. For instance the service SetDataSetValues is not exposed by the system due to the use of preconfigured data sets, thus the client is not able to set the values for the data sets. However, the system internally uses a method doing the same job as SetDataSetValues for creating the preconfigured data sets. Exposing this method would make it possible for the client to define its own data sets dynamically. Besides, the system exposes additional services not defined in [61400-25-3] such as services for retrieval of reporting and logging related data sets.

Mapping to web services

The system has been mapped to the web services mapping, which is one of the possible mappings defined by [61400-25-4]. The mapping has taken place on the

WCF platform by following guidelines for SOA. Platform specific data is kept within the service boundaries by use of the implicit data contract provided by WCF for simple data types such as strings and integers.

The system exposes its metadata in terms of WSDL which is used by the tool svcutil to generate a contract and configuration file for the system. These file are imported by the client in order to know which services the system exposes and how to communicate with the system in terms of address, binding, and contract (the "abc" for each endpoint in WCF). The generated contract and configuration file are specific for WCF. However the exposed metadata has also been used to generate the WSDL file which can be used to create a client on any platform capable of consuming web services. The WSDL file has not been used explicitly because both the client and system are implemented in WCF, hence the files generated by svcutil have been used instead.

The WSDualHttpBinding in WCF has been chosen because it provides a duplex channel on top of the HTTP protocol. The duplex channel makes it possible to use the publisher/subscriber approach for reporting, where the system is able to send spontaneous reports to the client.

Common control block related tasks such as managing subscriptions have been put in an abstract object named CB. Common reporting related tasks have been put into an abstract object named RCB. Unbuffered reporting, UBRCB, and buffered reporting, BRCB, inherit RCB which in turn inherits CB. Control block for logging, LCB, inherits CB directly. Putting reporting related tasks in RCB rather than in CB ensures, that LCB does not know anything about reporting.

Initialization

The system initializes the contents of its data model by reading the WPPCL file at startup. Only enabled data attributes are added to the data model.

Updating values of data model contents

At regular intervals, the system updates the values of its data model contents by use of the WPP data generator. The system loops through all its data attributes and for each one it asks for a value from the WPP data generator. The WPP data generator generates random data rather than realistic wind power plant data.

Using an updating interval of 60 seconds or less causes unexpected behavior with reporting in case of not established connection between system and client. The system uses 60 seconds for detecting that the connection is not established and if a new round of update starts before it is determined that the connection is not established, then more reports are trying to be sent and unexpected behavior is experienced. Two approaches for addressing this challenge can be adapted. Either the system must not use updating intervals with length less than 60 seconds, or a custom timer must be used that throws an exception before the next updating round is initiated.

Determining if reporting and logging must happen

When data model content values are updated, the system has an internal routine for determining if reporting and logging must occur. The outcome of the routine

depends on the data attribute that has been updated, its trigger option, the values before and after the update and if clients are subscribed to a data set that references this particular data attribute. Only the trigger dchg is used. The trigger dupd is not present in [61400-25-2], and regarding the trigger qchg, it is treated the same way as dchg.

Subscriptions

Each client is capable of configuring its own reporting and logging. When the client associates with the system, resources for reporting and logging for the particular client is allocated in terms of the objects UBRCB, BRCB and LCB, which has their own Subscription objects. By use of these resources, the client is able to express interest in reporting and logging in terms of subscriptions to data sets. When the client reconnects to the system after a disconnection, it uses its already allocated resources for reporting and logging, rather than getting new resources. This ensures, that the client keeps its subscriptions between connections. The control blocks stores the subscriptions in-memory, which is considered sufficient for a proof of concept model. However if the system is restarted, the subscription will be lost. Persistent storage for the subscriptions must be considered in order to address this challenge.

Reporting

When reporting happens, the system sends the report spontaneously to the client. Reporting happens, because a data set that the client has subscribed to references a data attribute that has been updated and its trigger condition been satisfied.

Two types of reporting are supported by the system: Unbuffered and buffered reporting. When the connection between system and client is established, the two types of reporting act similar. However, if the connection is not established, unbuffered reporting discards the report, and it is lost. Buffered reporting, on the other hand, buffers the report. The client can retrieve its buffered reports at next connection with the system.

Buffered reporting makes use of a unique report id for each report, increasing sequentially for new reports. This is used by the client to determine, if it has received all reports, or if some is missing. However, while the client is able to determine lost reports, it is not at the moment able to reissue a report with a given id. Each time the client reconnects to the system, it asks for buffered reports. If any, it asks the system to deliver them. The system delivers them all at once and then deletes them from the buffer. A finer grained solution for retrieving the buffered reports would be to let the client ask for the id's for the buffered reports. Then the client could ask for the reports in groups by id, for instance one at a time, or ten at a time. This approach would give the client control over the process for retrieving the buffered reports, thus balancing its load like the system balances its load by use of MinRequestTime and MaxRequestTime. This approach will make it possible for clients with relative few resources to retrieve buffered reports in balanced manner.

Regarding MinRequestTime and MaxRequestTime, the system uses them to determine the size of window, where reporting is activated. Reporting is only active in the window delimited by MinRequestTime and MaxRequestTime.

MinRequestTime is the time that has to elapse from a client has activated a report control block until it actually becomes active in the system and reporting can occur. At the other end, MaxRequestTime is the time that defines how long time a report control block can remain active before it is automatically deactivated by the system. The system informs the client when changes about the status for activation of RCB's happen. This is convenient for the client, because it does not control the process for activating and deactivating RCB's entirely by it self. In this thesis, the window size has been static. MinRequestTime has the value zero, that is, reporting is activated immediately when the client activates it. The value for MaxRequestTime is in order of minutes.

Logging

When logging happens, details about which data attribute caused the logging and the time it happened is logged to persistent storage, Access 2007 in this thesis. The log entries can later be retrieved by the client with filtering on time and entry id.

The system uses the same persistent storage for all clients. At the moment it is not possible to distinguish between log entries inserted by different clients. This can be solved in a straightforward manner by adding the unique client id as a field in the persistent storage. This will then be an implicit filter when clients retrieve the log entries in addition to the time and entry id filters.

Access Control

Although not an objective of the thesis, the concept of simple access control has been described and implemented. A simple all or nothing approach is used, that is, either the client is granted or denied access at the system level. In a real world scenario, the access control would be finer grained, for instance by specifying which parts of the data model the client is authorized to retrieve and use for reporting and logging.

6.1.2 Client

A client with SCADA has been created that is capable of consuming the services exposed by the system. The client imports the service model code generated by the tool svcutil based on the metadata exposed by the system. This provides the client with information about how to consume services from the system and which services to consume.

The client is capable of receiving reports from the system spontaneously, because it implements a point of contact and provides it to the system, as specified by the service model code.

The client supports both unbuffered and buffered reporting. When the connection between client and system is established, unbuffered and buffered reporting act similar, that is, reports are sent to the client spontaneously. Buffered reporting makes use of a unique report id for each report, which enables the client to verify, if it has received all reports, or if some is missing. When the client reconnects to the system, it retrieves its buffered reports by use of request-response, rather than the publisher/subscriber approach used for spontaneous reporting. This ensures that the client will be in control when retrieving the

buffered reports in terms of number of reports and the interval between requests. This approach enables clients will low resources to use the system safely.

A SCADA has been created for the client making it possible for human interaction with the client for using reporting and logging, and the system in general. The client has a unique id, which is used for identification by the system in the association process. This ensures that the client is delivered its own resources for reporting and logging. Besides, the client id has been used for implementing a simple access control in the system.

6.1.3 WPPCL file

The WPPCL file has been created as an xml file, representing the data model contents of a wind power plant. The contents of the WPPCL file consists of the hierarchical data with the server element in the top down to the data attribute element in the bottom.

6.1.4 WPPCL Editor

WPPCL Editor makes it possible to edit the WPPCL file. It provides a graphical user interface, making it intuitive to use. At startup, WPPCL Editor reads the current WPPCL file and presents the contents in a tree structure, similar to the tree in the SCADA. However, the tree in the WPPCL Editor shows all data attributes, both disabled and enabled. The tree in the SCADA only shows enabled data attributes due to the fact that only enabled data attributes are present in the system.

Coloring is used to enhance usability of WPPCL Editor. Enabled data attributes are green, and disabled data attributes are red. Use of the WPPCL is carried out by selecting a data attribute, then enabling or disabling it. This process is repeated until all relevant data attributes have been edited. When editing is over, WPPCL Editor saves the contents to the WPPCL file. At the moment, WPPCL Editor creates a new file rather than overwriting the old file.

6.2 Summary of Contributions

This thesis has demonstrated how the reporting and logging part of IEC 61400-25 can be applied in a real world software system. An operational IEC 61400-25 compliant system capable of reporting (unbuffered and buffered) and logging has been created. In order to consume the services exposed by the system, a client with SCADA has been created. The system is capable of modeling all IEC 61400-25 compatible wind power plants by use of the WPPCL file. The thesis has created the WPPCL Editor in order to configure the contents of the data model by editing the WPPCL file. Although the WPPCL file can be edited without WPPCL Editor, in terms of a simple text editor, WPPCL Editor provides an intuitive alternative to manual editing, and it does not assume that the user is familiar with xml. Besides, it is not error prone like the manual editing.

By demonstrating that IEC 61400-25 can be used to construct a working system in real world software, the intention of this thesis is that future modelers of wind power plants in the software domain will adapt IEC 61400-25. This

will make it possible for actors in the wind power plant market to cooperate based on business related parameters rather than being constrained by different approaches for modeling wind power plants. Ideally, plug and play approach will be within reach by use of IEC 61400-25. The effect of this will be a more dynamic wind power plant market with increased degree of readiness to change. This will trim the wind power plant sector into a more competitive sector compared to external competition, such as the oil industry. When this is achieved, IEC 61400-25 has done its part for creating a brighter future for wind power plants.

6.3 Discussion and Future Work

6.3.1 IEC 61400-25 compliant system

- Modeling of logical nodes: The modeled information model in the system has been selective with regards to the information model defined in [61400-25-2]. For instance the structure for logical nodes are inherited from and defined in [61850-7-2] (clause 9.1.1) where it can be seen that each logical node is a container for zero or multiple data sets, report control blocks and log control blocks besides the data. The information model used by the system in this thesis has only used the data aspect for logical nodes. Data sets and control blocks have not been a part of the logical nodes. However they have been part of the information model, but separate from the logical nodes. What would have been different if these elements were part of the logical node? A more distributed information model would be reality, where new control blocks would have to be added to logical nodes. By separating control blocks from logical nodes and modeling them separately in the information model, reporting and logging is separated from the contents of the data model, thus achieving low coupling, in pattern terms, of unrelated tasks.

  Regarding modeling of the information model, it must be noted that LLNO and LHPD from [61400-25-2] has not been considered in this proof of concept model.

- Modeling of control blocks: The control blocks for reporting and logging does not reflect all contents as defined in [61850-7-2]. The subscriptions has been given relative higher priority while modeling the control blocks. For instance the BRCB in [61850-7-2] has field for indicating buffer overflows. Such details has not been used by the BCRB in this thesis. However the control blocks are well defined in the information model and future work for improving them would be focused on improving the respective control block classes, thus it would be a well defined task.

- MinRequestTime and MaxRequestTime: In this thesis, the size of window determined by MinRequestTime and MaxRequestTime has been static. It would be more convenient to let the system determine the size of window dynamically based on the load in the system. This implies that resources in the system such as memory must be monitored.

- Retrieving buffered reports in groups: The client should be able to retrieve its buffered reports in groups rather than retrieving all reports at one time,

as is the case at the current moment. If the client is able to control the size of groups with reports that are retrieved, it will be able to balance load in accordance with its resources. This approach will ensure, that even relative resource weak devices such as handheld devices will be able to use the system safely.

- Multiple clients: In order to use the system in an environment with multiple clients, challenges with concurrency must be addressed. This applies to the business logic for the system, such as creating thread safe access to the Singleton objects, and it applies to the handling of the incoming connections from clients. Related to multiple clients, logging should make use of the unique client id when inserting and retrieving log entries. This is because log entries for all clients are inserted to the same persistent storage.
- Data attribute: Properties for data attributes should be typed in accordance with [61400-25-2]. Data attributes from [61850-7-2] should be typed in as well.
- Storing subscriptions: Currently, subscriptions for reporting and logging are stored in memory. When the system is restarted all subscriptions will be lost. Persistent storage for the subscriptions should be considered.
- Access control: Although access control has not been a subject for the thesis, a simple approach has been proposed and implemented. It is an all or nothing approach, that is, access is granted or denied at the system level. A finer grained approach would be to specify the access control for different parts of the contents of the data model.
- Mandate: IEC 61400-25 defines mandatory elements and optional elements, also for the data model contents. By providing the system with these rules and applying them, the system would be able to determine if a given WPPCL file conforms to the rules. If not, the system could notify the client, that the WPPCL was not truly IEC 61400-25 compliant.
- Hosting: Currently, the system is hosted with the self hosting approach, that is, the hosting is created by the developer as part of the application. Hosting on WAS should be considered before deployment of system on large scale in order to take advantage of built in features in WAS such as reliability and connection pooling. The business logic currently in the system can be reused when changing the type of host. IIS could also be considered. However WAS is capable of doing all that IIS can do plus more. If an option, WAS should be preferred.
- Dynamic data sets: Dynamic data sets could be considered, that is, the client should be able to set the rules for data sets in the lifetime of the system. However in such case it must be analyzed if data sets must be on per client basis rather than on per system basis as it is the case currently.

6.3.2 Client

If implemented by the system, retrieval of buffered reports in groups should be able to be consumed by the client. Either the size of groups could be determined

dynamically by the client based on its resources or it could be decided by the user, through the SCADA. Maybe a mix could be used.

The contents of the data model is visualized in the tree structure in the SCADA. It could be a convenient feature to be able to click on a data attribute in the tree and choose to retrieve details such as functional constraint, description and other data attribute properties as well as the value for the data attribute. Properties and the value for the data attributes are already within the system. Retrieval of this information is related to retrieval of data model contents. The service GetDataValues (as presented in the information exchange model, but not used) must be mapped and exposed in order for the client to retrieve data attribute properties and values.

6.3.3 WPP Data Generator

The WPP data generator used in this thesis has been a simple random data generator. This part is a candidate for improvement. It can be achieved by either making WPP data generator generate wind power plant realistic data or by use of real wind power plants. This thesis proposes, that real wind power plants are used, because eventually the system must be used with real wind power plants, this is the purpose of IEC 61400-25. This part must be done with people that has insight and knowledge in wind power plants. The interface between system and wind power plant must be modified if necessary depending on the actual wind power plant. Actually this interface could be created in the WPP data generator object, and the rest of the system will not notice any difference from the existing solution. However, this assumes that the real wind power plant can be polled for data like the existing WPP data generator is capable of. These topics are out of scope for the writer and will be left open for future research and work.

6.3.4 WPPCL file

If access control is used at the level of contents in the data model, this could possibly be reflected by the WPPCL file. Like the field that indicates if a data attribute is enabled or disabled, a new field could be introduced to indicate which groups of users that have access to the data model contents. The access could be defined on the level of logical nodes. Then users with responsibility or knowledge about for instance the wind power plant turbine would be grouped as ”groupWTUR” and this field added to the logical node ”WTUR” in the WPPCL file. Access control could also be defined at logical device level. This would enable the system to service clients from different companies, without letting them access wind power plants from other companies than their own. Each client would be a member of one or multiple user groups, defining which parts of the data model it has access to. This would also work with reporting and logging by checking the groups the client belongs to before doing reporting and logging.

6.3.5 WPPCL Editor

The WPPCL Editor should be refined in order to make it is possible to have editing at all levels of the data model rather than only at the level of data

attributes. If access control is adapted by the WPPCL file, the WPPCL Editor should also be able to cope with that besides enabling and disabling data attributes.

Regarding the mandatory and optional elements defined by IEC 61400-25 in terms of data model contents, WPPCL Editor could also have the responsibility of verifying if a given WPPCL file was truly compliant with the standard, that is, if it included all the mandatory elements as a minimum. A ”compliant mode” could be introduced for WPPCL Editor, where it would not be possible by accident to disable mandatory elements. If for some reason this was the intention, ”compliant mode” could be disabled and the user would have all control again.

Bibliography

[Andreas & Baris] Andreas and Baris, *Prototype for a IEC 61400-25 Compliant Generic Server*, IMM, 2006.

[61400-25-1] IEC, *Wind Turbines - Part 25-1: Communications for monitoring and control of wind power plants - Overall description of principles and models*

[61400-25-2] IEC, *Wind Turbines - Part 25-2: Communications for monitoring and control of wind power plants - Information models*

[61400-25-3] IEC, *Wind Turbines - Part 25-3: Communications for monitoring and control of wind power plants - Information exchange models*

[61400-25-4] IEC, *Wind Turbines - Part 25-4: Communications for monitoring and control of wind power plants - Mapping to communication profile*

[61400-25-5] IEC, *Wind Turbines - Part 25-5: Communications for monitoring and control of wind power plants - Conformance testing*

[61850-7-2] IEC, it IEC 61850-7-2 Basic communication structure for substation and feeder equipment - ACSI

[Löwy] Juval Löwy, *Programming WCF Services*, Copyright 2007 O'Reilly Media

[Larman] Craig Larman, *Applying UML and Patterns: An introduction to object-oriented analysis and design and the Unified Process*, Second edition, 2002 Prentice Hall

[Harikumar] Kumar Harikumar, *An Event Driven Architecture for Application Integration using Web Services*, p. 542-547. In: Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Confere...., A. 2005

[Elforsk] Anders Johnson, Jörgen Svensson, *Wind power communication - Design and implementation of test environment for IEC61850/UCA2*, April 2002

[WS-Eventing] *Web Services Eventing (WS-Eventing)*, Microsoft

Appendices

Appendix A

Data Sets

A.1 WSLG

Data sets for logging of states (WSLG) consist of

A.1.1 TurCmLog

CMD actSt[CO], actSt[ST]

A.1.2 TurStLog

STV actSt[ST]

A.1.3 HiUrgAlm

ALM almAck[CO], actSt[ST]

A.1.4 LoUrgAlm

ALM almAck[CO], actSt[ST]

A.1.5 TurCtLog

CTE actCtVal[ST], ctTot, dly, mly, yly

A.1.6 TurTmLog

TMS actTmVal[ST], tmTot, dly, mly, yly

A.2 WALG

Data sets for logging of states (WALG) consist of

A.2.1 TurAnLog

MV mag, range, q

A.2.2 TurPhLog

WYE cVal, range, q

A.2.3 HiAcsSp

SPV actVal[CO], actVal[ST], incRate, decRate, minVal, maxVal

A.2.4 LoAcsSp

SPV actVal[CO], actVal[ST], incRate, decRate, minVal, maxVal

A.2.5 TrgEmStop

MV mag, range, q. SPV actVal[CO], actVal[MX], incRate, decRate, minVal, maxVal. STV actSt[ST]. ALM almAck[CO], actSt[ST]. CMD actSt[CO], actSt[ST].

A.2.6 TrgProdGri

MV mag, range, q. SPV actVal[CO], actVal[MX], incRate, decRate, minVal, maxVal. STV actSt[ST]. ALM almAck[CO], actSt[ST]. CMD actSt[CO], actSt[ST].

Appendix B

Source code

The source code for the IEC 61400-25 compliant system, the client and the WPPCL Editor is all in the provided cd-rom. This approach has been chosen, because the layout and structure of the source code is not suited to be printed on paper. It could be done, but readability would be reduced greatly.

B.1 IEC 61400-25 compliant system

Refer to the folder ”System” on the cd-rom. The Access database used for logging is provided in the subfolder ”database”.

B.2 Client

Refer to the folder ”Client” on the cd-rom

B.3 WPPCL Editor

Refer to the folder ”WPPCL Editor” on the cd-rom

Appendix C

WPPCL file

Like the source code, the WPPCL file has been placed in the provided cd-rom, rather than being printed on paper. The WPPCL file can be found in the folder ”WPPCL file” on the cd-rom.

Appendix D

WSDL file

The WSDL file has been generated by the tool disco.exe from the metadata exposed by the system. Refer to the folder named ”WSDL” on the cd-rom.