UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

AUTOMATION TECHNOLOGY

Christian Söderbacka

THE GOOSE PROTOCOL

Master´s thesis for the degree of Master of Science in Technology submitted for evaluation, Vaasa, 15 March 2013.

Supervisor	Jarmo Alander
Instructor	Petri Välisuo, Leif Strandberg

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Ralf Christian Söderbacka
Diplomityön nimi: GOOSE-protokolla
Valvojan nimi: Professori Jarmo Alander
Ohjaajan nimi: TkT Petri Välisuo, DI Leif Strandberg
Tutkinto: Diplomi-insinööri
Yksikkö: Sähkö- ja energiatekniikan yksikkö
Koulutusohjelma: Sähkö- ja energiatekniikan koulutusohjelma
Suunta: Automaatiotekniikka
Opintojen aloitusvuosi: 2009
Diplomityön valmistumisvuosi: 2013 **Sivumäärä: 118**

TIIVISTELMÄ

Merkittävä osuus uusista sähköasemista ympäri maailman noudattaa kansainvälistä IEC 61850 -sähköasemastandardia, joka määrittelee yhdenmukaisesti tietoliikennekäytännöt niiden elektronisten laitteiden välillä, joista sähköaseman automaatiojärjestelmä koostuu. Standardin menestys kasvattaa sen suosiota myös muissa sovelluksissa, ja sitä hyödynnetäänkin esimerkiksi tuulivoimaloissa, vesivoimalaitoksissa ja älykkäissä sähköverkoissa. Tämän diplomityön kannalta kiinnostavinta on IEC 61850 sovellettuna polttomoottoreiden ohjausjärjestelmän ja voimalaitoksen oheislaitteiden väliseen kommunikointiin.

Tämän diplomityön tavoitteena on kerätä kokemusta IEC 61850 -standardin sisältämän GOOSE -kommunikointiprotokollan käytöstä. Työssä esitellään standardia ja siihen liittyviä käsitteitä yksinkertaisten esimerkkien avulla. Lisäksi on kehitetty Linux-ympäristössä toimiva, avoimeen lähdekoodiin perustuva ohjelma, jonka avulla voidaan lähettää GOOSE -viestejä kahden eri laitteen välillä. Työn pääpaino on siis GOOSE-protokollan ja kehitetyn ohjelman esittelyssä.

Tulokset osoittavat, että GOOSE -protokolla mahdollistaa yhteensopivan tiedonvaihdon kahden eri valmistajan laitteiden välillä, joten sitä voidaan pitää varteenotettavana vaihtoehtona lähitulevaisuuden kommunikointiprotokollaksi. IEC 61850 -standardia saatetaan hyvinkin ottaa lähitulevaisuudessa käyttöön polttomoottorivoimaloissa. Käyttöönoton aikataulu riippuu osittain markkinoiden kehityksestä sekä asiakkaiden asettamista vaatimuksista.

AVAINSANAT: IEC 61850, GOOSE, SAS, DER

UNIVERSITY OF VAASA
Faculty of Technology
Author: Ralf Christian Söderbacka
Topic of the Thesis: The GOOSE Protocol
Supervisor: Professor Jarmo Alander
Instructor: Ph.D. Petri Välisuo, M.Sc. Leif Strandberg
Degree: Master of Science in Technology
Department: Department of Electrical Engineering and Energy Technology
Degree Programme: Degree Programme in Electrical and Energy Engineering
Major of Subject: Automation Engineering
Year of Entering the University: 2009
Year of Completing the Thesis: 2013 **Pages: 118**

ABSTRACT

The majority of the electrical substations built today conform to the international standard IEC 61850 that uniformly defines the communication between the various intelligent electronic devices (IEDs) of the substation automation system. The success of the standard in substation automation has expanded its application to new areas, such as wind power, hydro power, and smart grids. For this thesis, the most intriguing use of the standard is its application to the communication between the control system of internal combustion engines and other equipment in a power plant.

The main objective of this thesis is to gain early experience on the use of the Generic Object Oriented Substation Events (GOOSE) protocol, which is often considered the most prominent communication protocol of IEC 61850. In this thesis, the standard IEC 61850 is described, and its most fundamental concepts are illustrated by clear examples. An application based on open source software has been developed in the Linux-environment in conjunction with this thesis. The application enables two IEDs from different manufacturers to exchange GOOSE messages. The emphasis of the thesis is thus on the presentation of the GOOSE protocol and the developed application.

The results show that the GOOSE protocol provides compatible interfaces for information exchange between IEDs provided by different manufacturers, and can be seen as a viable option as a future communication protocol. IEC 61850 will likely be put into service in engine power plants, in the near future. The schedule partially depends on the development of the markets and the requirements of the customers.

KEYWORDS: IEC 61850, GOOSE, SAS, DER

VASA UNIVERSITET
Tekniska fakulteten
Författare: Ralf Christian Söderbacka
Diplomarbetets titel: GOOSE-protokollet
Övervakare: Professor Jarmo Alander
Handledare: TkD Petri Välisuo, DI Leif Strandberg
Examen: Diplomingenjör
Enhet: Enheten för elektro- och energiteknik
Utbildningsprogram: Utbildningsprogrammet för elektro- och energiteknik
Inriktning: Automationsteknik
Årtal för inledande av studier: 2009
Diplomarbetet färdigställt: 2013 **Sidantal: 118**

ABSTRAKT

Majoriteten av de elstationer som byggs idag tillämpar den internationella standarden IEC 61850 som enhetligt definierar kommunikationen mellan de intelligenta elektroniska apparater som utgör elstationens automationssystem. Eftersom IEC 61850 har varit en framgångssaga i global skala ökar dess popularitet också inom alternativa användningsområden, till exempel vindkraft, vattenkraft och smarta elnät. För detta examensarbete ligger den mest intressanta tillämpningen av standarden i kommunikationen mellan styrsystemet för förbränningsmotorer och övrig utrustning i ett motorkraftverk.

Målet med detta examensarbete är att samla erfarenhet kring tillämpningen av GOOSE -protokollet, som ofta anses vara det viktigaste enskilda kommunikationsprotokollet i IEC 61850. I arbetet beskrivs standarden och dess koncept demonstreras med enkla exempel. En applikation baserad på öppen källkod har utvecklats i Linux-miljö i samband med detta examensarbete. Applikationen möjliggör kommunikation i form av GOOSE -meddelanden mellan två olika elektroniska apparater. Tyngdpunkten i examensarbetet ligger således på att beskriva GOOSE-protokollet och den utvecklade applikationen.

Resultaten visar att GOOSE -protokollet tillåter kompatibelt informationsutbyte mellan olika elektroniska apparater från olika tillverkare, och kan anses som ett hållbart alternativ som framtidens kommunikationsprotokoll. IEC 61850 kommer sannolikt att tas i bruk i motorkraftverk i den närmaste framtiden. Tidtabellen för ibruktagningen beror delvist på marknadens utveckling samt på de krav som ställs av kunderna.

NYCKELORD: IEC 61850, GOOSE, SAS, DER

TABLE OF CONTENTS **page**

SYMBOLS AND ABBREVIATIONS

$	Object name separation mark used by the MMS protocol
ACSI	Abstract Communication Service Interface
CDC	Common Data Class
CID	Configured IED Description
DER	Distributed Energy Resources
EMF	Eclipse Modelling Framework
FC	Functional Constraint
FCD	Functional Constrained Data
FCDA	Functional Constrained Data Attribute
GCB	GOOSE Control Block
GoCB	GOOSE Control Block
GOOSE	Generic Object Oriented Substation Events
GSE	Generic Substation Event
I/O	Input / Output
ICD	IED Capability Description
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Device
IID	Instantiated IED Description
IP	Internet Protocol
ISO	International Organization for Standardization
JET	Java Emitter Templates
LAN	Local Area Network
LDU	Local Display Unit
LLN0	Logical Node zero
LN	Logical Node
LN0	Logical Node zero
LPHD	Logical node physical device
MAC	Media Access Control
Mbps	Mega bits per second
MCAA	Multicast Application Association

MMS	Manufacturing Message Specification
OSI	Open System Interconnection
PTPv2	Precision Time Protocol
q	Quality
SAP	Service Access Point
SAS	Substation Automation System
SCD	System Configuration Description
SCL	System Configuration Language
SCSM	Specific Communication Service Mapping
SNTP	Simple Network Time Protocol
SSD	System Specification Description
SV	Sampled Values
t	Timestamp
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TPAA	Two-party Application Association
TrgOp	Trigger Option
VLAN	Virtual Local Area Network
VMD	Virtual Manufacturing Device
XCBR	Circuit Breaker logical node class
XML	eXtensible Markup Language

ACKNOWLEDGEMENTS

This Master's thesis was initiated by Wärtsilä Finland – PowerTech Research and Development, Department of Automation and Control. It concerns the implementation of the standard IEC 61850 for communication in power plants.

First, I would like to thank my instructor at Wärtsilä, M.Sc. Leif Strandberg, and his superior, M.Sc. Pasi Juppo, for giving me the opportunity to write this thesis. I'd also like to thank my supervisor, Professor Jarmo Alander, and my second instructor, Ph.D. Petri Välisuo, at the University of Vaasa, for their guidance and helpful advice. Very special thanks go to my co-worker at Wärtsilä, M.Sc. Staffan Tunis, for continuously advancing the software development, and for proofreading of several drafts of my thesis. Yet another person I'd like to thank is M.Sc. Olavi Vähämäki, R&D Director at Vamp Ltd., for letting me borrow a Vamp 50 protection relay, and for providing me with multiple useful documents.

Last but not least, I'd like to thank my fellow board members at Wasa Teknologförening[1] (WTF), for making this final year of study a memorable one.

Vaasa 11.3.2013

Christian Söderbacka

[1] A society founded in 2012 for Swedish-speaking Master's students at the University of Vaasa and Åbo Akademi.

1. INTRODUCTION

Electrical substations often have protection and control devices that have been produced by a variety of different manufacturers. Traditionally, different devices have used different types of communication protocols, making them inherently incompatible in networking. (IEC 61850-1 2003: 9.)

The standard IEC 61850 presented by the International Electrotechnical Commission (IEC) was designed to standardize the communication between different intelligent electronic devices (IEDs). This has been done mainly by defining rules on how data should be modeled and organized in a way that is consistent across different devices. The standardization brings compatibility between different kinds of devices provided by different manufacturers and reduces the engineering effort required to configure the substation automation system (SAS). Although many modern electrical substations are built to conform to IEC 61850, the standard is also being used extensively in other applications, for example in wind and hydro power systems. (IEC 61850-1 2003: 9-10; Mackiewicz 2006.)

This Master's thesis was initiated by Wärtsilä Finland, Department of Automation and Control, as a part of a research project set to investigate the possibilities of integrating the standard IEC 61850 into the embedded electronic devices (modules) used for the control of *reciprocating engines*[2]. The goal is to get a more direct communication between the engine control system, which utilizes a proprietary protocol, and the supplementary systems, which already implement the Generic Object Oriented Substation Events (GOOSE) protocol.

IEC 61850 specifies a set of *logical nodes* for the modeling of different functions in a SAS. Logical nodes suitable for the modeling of a reciprocating engine can be found in an extension to the standard, the IEC 61850-7-420, which contains specifications of logical nodes for the Distributed Energy Resources (DER) domain.

[2] Internal combustion engines running on for example methane gas or diesel.

In this thesis the IEC 61850 standard is studied, and the concepts are demonstrated using simple examples where possible. The material covered in this thesis is based on scientific research articles and on the standard itself. Several earlier theses dealing with similar topics were also studied.

A simple application based on an open source platform was developed and run on one of Wärtsilä's proprietary modules called LDU, which is short for Local Display Unit 20. The application enables the LDU to transmit and receive GOOSE messages over an Ethernet local area network (LAN). To verify that the LDU is capable of communicating with another IED from a different manufacturer, an interoperability test was set up where GOOSE messages were exchanged between the LDU and a Vamp 50 protection relay provided by Vamp Ltd., Vaasa, Finland. The network traffic was monitored using the Wireshark Network Analyzer tool, and PuTTY, which is an SSH and telnet client.

The results confirm that the GOOSE protocol can indeed be used for communication between IEDs provided by different manufacturers. The successful interoperability test justifies further research and development—the ultimate goal is to exploit the full potential of IEC 61850 in certain modules of the engine control system.

This thesis, apart from the introduction, is divided into four chapters. Chapter 2 presents IEC 61850 as it applies to electrical substation automation. It introduces multiple standard-specific notions and serves to give the reader a basic understanding of IEC 61850. Chapter 3 presents the practical work done in this thesis, where an application implementing GOOSE messaging was developed and tested. The hardware, software tools, and code development is briefly described. Chapter 4 presents the *Abstract Communication Service Interface* (ACSI), used for modeling the substation devices, the data to be exchanged between devices, and the services used for reading and writing attribute values in devices. The chapter also presents the *Specific Communication Service Mappings* (SCSMs), needed to map the abstract Application layer protocols of IEC 61850 to a concrete transport protocol (Ethernet). Chapter 5 presents the conclusions. It also brings up the difficulties that were met during the study of the standard, and concludes with a discussion of conformance testing.

2. IEC 61850 IN SUBSTATION AUTOMATION

The international standard IEC 61850 ‘Communication networks and systems for power utility automation’[3] consists of multiple different parts, and makes use of a variety of existing standards. Several other documents are also used in conjunction with the standard, like the extension IEC 61850-7-420 ‘Basic communication structure – Distributed energy resources logical nodes', the IEC 61850-9-2LE ‘Implementation guidelines for digital interface to instrument transformers using IEC 61850-9-2’, the IEC 62439-3 ‘High availability automation networks’[4], and the IEC 61588 ‘Precision clock synchronization protocol for networked measurement and control systems’[5]. (IEC 61850-1 2003: 10.)

An SAS can be described as “a supervisory management and control system for industrial electrical distribution systems.” The SAS is implemented using a number of IEDs to perform the required functions, such as monitoring, protection, control, and data processing. ‘IED’ is a collective name for microprocessor-based devices with networking capabilities. An IED can be described as: “an instrumentation and control device that is capable of collecting and reacting to data and then use this data to create information.” Examples of common types of IEDs are protection relays, circuit breaker controllers and voltage regulators. To realize communication between different IEDs, a communication protocol is required. A protocol can be described as “a set of rules that must be obeyed for orderly communication between two or more parties.” For the communication to work, the IEDs need to be interoperable. Interoperability refers to the ability of devices to operate on the same network or communication path sharing the same information and commands. (Ozansoy 2006: 1, 13, 17; IEC 61850-1 2003: 9.)

---

[3] The first edition of IEC 61850 was known as ‘Communication networks and systems in substations’ but the name was changed in the second edition due to the expanded use of the standard.

[4] Not considered in this thesis.

[5] Corresponds to IEEE 1588.

2.1. The Open System Interconnection model

The Open System Interconnection (OSI) 7-layer reference model, depicted in Figure 1, is based upon the concept of layering of communication functions, and is frequently used as a framework for describing how a layered protocol stack operates. Each layer in the reference model is responsible for specific tasks, and uses services offered by the layer below while providing services to the layer above. The communication between two adjacent layers is called an *interface*. The higher up in the stack a layer resides, the more abstract view it has on the lower layer implementation details. This improves the interoperability between the layers and between different network devices using different platforms and operating systems. It also facilitates network programming. OSI also recognizes the existence of application profiles (layers 5-7), and transport profiles (layers 1-4) in the OSI model. Different combinations of application profiles and transport profiles are used for the transmission of different types of messages. The profile combinations used for client/server and GOOSE (publisher/subscriber) messaging can be found in Appendix 1. Sockets are, in the UNIX domain, interfaces from the upper three layers into the Transport layer. (Scaglia 2007: 11-12; IEC 61850-8-1 2011: 25; Stevens 1998: 18.)

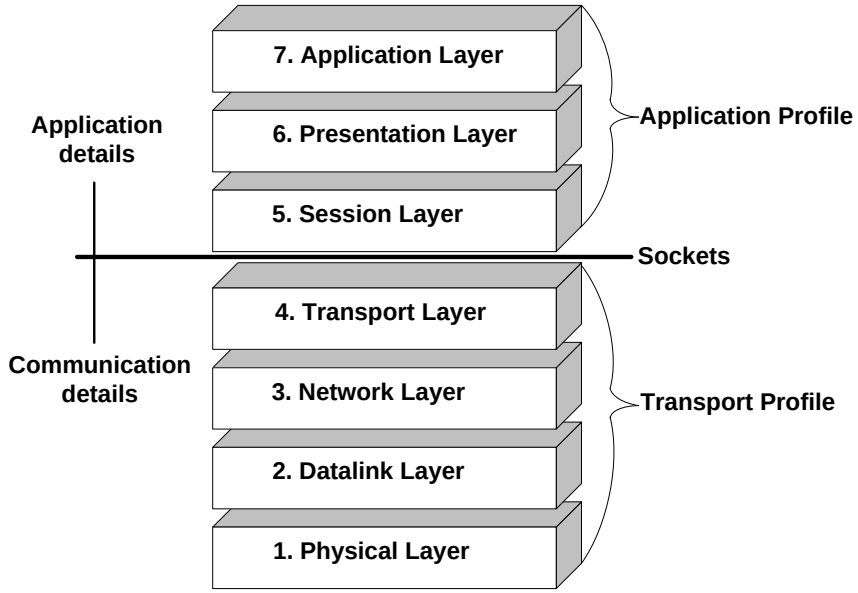


Figure 1. The layered OSI reference model. (Adapted from Stevens 1998: 18.)

2.2. Networking in substation automation

The functional hierarchy of a substation can be conceptually divided into three different levels: the station, bay, and process level. Figure 2 depicts the functional hierarchy the way it was perceived by the author (the figure is actually a compilation of several different figures from different sources). Application functions of the SAS can be distributed between IEDs on the same, or at different levels of the functional hierarchy. The free allocation of functions to IEDs enables different approaches in function integration, function distribution, and substation automation architecture. (IEC 61850-1 2003: 12-14; Yashwant & Swarup 2011.)

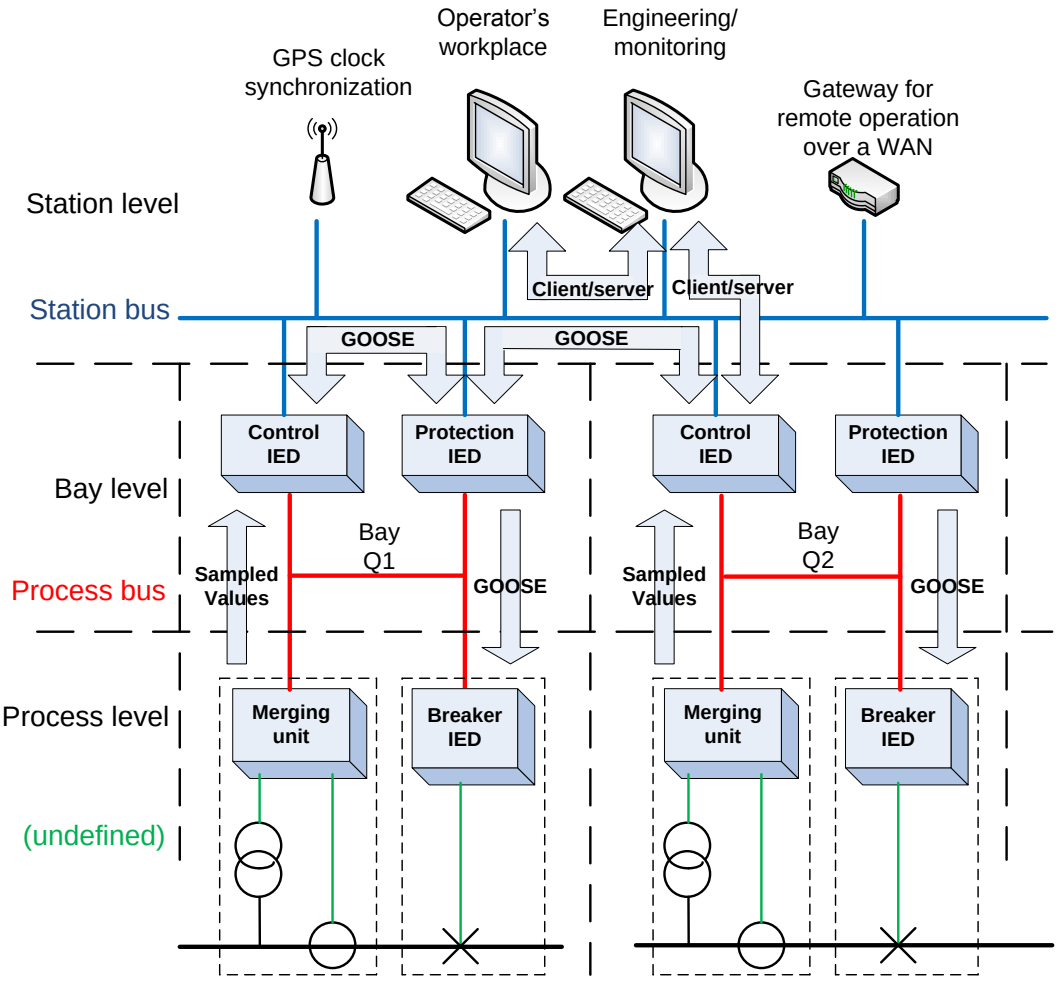


Figure 2. Conceptual model of a distributed protection system of an electrical substation conforming to IEC 61850. The arrows show where the different message types of IEC 61850 travels. (Adapted from Schnakofsky 2011: 16.)

The process level relates to collecting data and status information from the *primary equipment* (e.g. instrument transformers, circuit breakers etc.) in a bay, as well as to operating on the primary equipment, e.g. tripping of circuit breakers, control of disconnecting switches etc. (Janssen & Apostolov 2008; Xyngi & Popov 2010.)

The bay level consists of separate bays each with control, protection and monitoring devices, i.e., *secondary equipment*. A 'bay' is a collective name for a power system subset consisting of closely connected subparts with some common functionality, like a circuit breaker and its associated isolators, earth switches, and instrument transformers, or a transformer with its related switchgear between two busbars representing two different voltage levels. Each (medium voltage) feeder cubicle inside the substation control house typically counts as a bay, where the control and protection functions are typically performed by the same IED. In a high voltage bay, for example in the switchyard, the control and protection functions are typically performed by different IEDs—the controller IED being called "bay controller". (IEC 61850-1 2003: 7, 14; Ingram, Schaub & Campbell 2011; Starck 2012.)

The station level is the station supervisory level that holds the station computer, which contains among other things a human-machine interface (HMI), i.e., a Supervisory Control and Data Acquisition (SCADA) system, with which the operator can monitor and control the substation. The station level generally also has interfaces for remote communication and Global Positioning System (GPS) clock synchronization. (IEC 61850-1 2003: 14.)

The station bus and the process bus (see Figure 2) are key concepts in IEC 61850. They are defined by IEC 61850-8-1 and IEC 61850-9-2, respectively. The station bus and the process bus are usually perceived as two physically separate networks, although it is possible to have them share the same network infrastructure, i.e., an Ethernet LAN. Both networks can therefore be used to transmit IEC 61850-specific message types like

- time-critical connectionless multicast data stream of Sampled Values (SV) mapped directly onto Ethernet frames on the DataLink layer,

- time-critical connectionless multicast GOOSE messages mapped directly onto Ethernet frames on the DataLink layer,
- non-time-critical, connection-oriented, unicast client/server messages mapped to the Manufacturing Message Specification (MMS) protocol that operates over the Transmission Control Protocol / Internet Protocol (TCP/IP) and Ethernet stack.

The Simple Network Time Protocol (SNTP) and Precision Time Protocol (PTPv2) time synchronisation protocols are typically used to provide the time synchronization in substations conforming to IEC 61850. SNTP can be used where high precision is not required, and is sometimes used in the station bus. PTPv2 is capable of the sub-microsecond precision required by GOOSE and SV messaging, and is therefore used in the process bus. Time synchronization is left outside of this thesis. (McGhee & Goraj 2010; Ingram, Schaub & Campbell 2012; IEC 61850-9-2 2011: 16.)

The different message types are typically assigned different VLAN (Virtual LAN) identifiers and priority levels. VLAN allows an Ethernet switch to deliver information only to those switch ports/IEDs that have subscribed to the data. The time-critical GOOSE and SV messages are expected to deliver data fast and reliably and should therefore be assigned a higher priority than non-time-critical messages. High priority messages get processed ahead of low priority messages in switch queues. (Ingram, Schaub & Campbell 2011; Mackiewicz 2006; Xyngi & Popov 2010.)

A substation usually has one global station bus but multiple process buses, one for each bay. As depicted in Figure 2, the IEDs residing at the bay level are simultaneously connected to both the station bus and the process bus via independent network interfaces. The process and station busses are typically realized as Fast Ethernet (100 Mbps) full-duplex fibre optic Ethernet LANs. The process bus might have to be realized as Gigabit Ethernet (1000 Mbps or more) to accommodate the network traffic. The station bus is used for communication between

- different devices residing at the station level,
- devices residing at the station level and devices residing at the bay level (vertical communication), and

- devices residing in different bays (horizontal communication).

The process bus interconnects all IEDs within a bay, and is used for the communication between devices residing at the bay and process levels, e.g. between the primary and secondary equipment. (Liang & Campbell 2008; Ingram, Schaub & Campbell 2011; Janssen & Apostolov 2008; Xyngi & Popov 2010; De Mesmaeker, Rietmann, Brand & Reinhardt 2005; McGhee & Goraj 2010; Moore, Midence & Goraj 2010; Moore & Goraj 2011.)

A *merging unit* (see Figure 2) is an IED residing at the process level that collects data from both conventional (passive) and non-conventional (microprocessor-based) current transformers and voltage transformers, as well as binary status information from the I/O units of primary devices such as circuit breakers and other switchgear. It also acquires analogue values from other transducers and sensors. The merging unit need not necessarily be a standalone unit as it can be integrated into non-conventional instrument transformers. (Moore & Goraj 2011; IEC 61850-9-1 2003: 29.)

The merging unit in a 50 Hz system continuously samples the input values at a rate of 80 samples per cycle for protection applications and 256 samples per cycle for power quality monitoring and waveform recording applications, as specified by the IEC 61850-9-2LE guideline document. The Sampled Values receive a timestamp and are transmitted as a continuous stream over the process bus. The merging unit transmits 4000 messages per second, which results in a traffic rate of 4.4 Mbps. (Moore & Goraj 2011; Ingram, Schaub & Campbell 2011.)

The process bus conveys the SVs from the process level to the bay level, and GOOSE messages from the bay level to the process level. The SV messages are processed by the subscribing protection and control devices, which can then take appropriate actions like sending GOOSE messages to trip a breaker. The digitized SVs must be received fast, in synchronism, and without any interruptions by the protection devices in order for the protection algorithms to function properly. (Ingram, Schaub & Campbell 2011; Moore & Goraj 2011; Janssen & Apostolov 2008.)

The station bus conveys non-time-critical messages by the client/server communication profile between devices at the station level, and between devices at the station and bay levels. To reach devices at the process level, the messages are forwarded over the process bus. Therefore supervision and maintenance of the devices connected to the process bus can be handled from the HMI and the substation gateway connected to the station bus. The station bus also conveys time-critical GOOSE messages with critical protection information between bays, nominally at a 4 kHz sample rate. The GOOSE message contents can be binary (e.g. interlocking or blocking), or transduced analogue values (e.g. measurement information.) (Moore & Goraj 2011; Ingram, Schaub & Campbell 2011; Kirrmann, Rietmann & Kunsmann 2008; Antonova, Frisk & Tournier 2011.)

Both SV and GOOSE messages are published as multicast messages. The multicast publishing model is connectionless, which means that the publisher will not receive any acknowledgement of a successfully delivered message from the subscribing devices. However, multicasting of messages is efficient and enables high levels of real-time traffic over the process bus. The SVs are normally not allowed to enter the station bus as they consume bandwidth and load IEDs that do not require receiving the SV stream. The client/server communication utilizes the TCP/IP protocol suite, which guarantees the delivery of unicast messages. These messages might be for instance settings and file transfers. (Antonova, Frisk & Tournier 2011; Ingram, Schaub & Campbell 2011.)

High levels of multicast message traffic might flood the network and affect the performance of protection IEDs and PTPv2 clocks. If the level of traffic exceeds the network capacity or the capacity of the subscribers to process messages, multicast address filtering can be used in the Ethernet switches. Multicast address filtering restricts the SV and GOOSE traffic to defined subsets of subscribers. The switches need to ensure that the traffic rate does not exceed 100 Mbps as this is the maximum rate that the links and the Fast Ethernet interfaces of the IEDs can manage without degradation in performance. (Ingram, Schaub & Campbell 2011.)

GOOSE messaging enables simple testing and simulation of the SAS as any GOOSE message can be monitored and reproduced using a laptop PC, as illustrated in Figure 3.

Complicated and substation-wide virtual testing such as breaker fail trip can be performed without setting up any temporary wiring. The laptop can be used to send GOOSE messages to the system, and to monitor the results via GOOSE or MMS message reports. (Bekker, Diamandis & Tibbals 2010.)

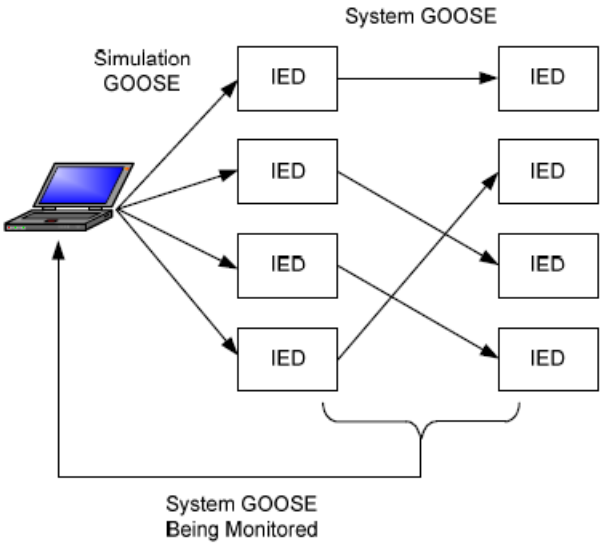


Figure 3. Laptop sending simulation GOOSE messages and monitoring system GOOSE messages during the testing of an SAS. (Bekker, Diamandis & Tibbals 2010.)

2.3. Interoperability and interchangeability

Part 7-1 of the standard states the following: "The goal of the IEC 61850 series is to provide interoperability between the IEDs from different suppliers or, more precisely, between functions to be performed by systems for power utility automation but residing in equipment from different suppliers." Interoperability refers to the ability of the interfaces of two different systems to cooperate—what this means in practice is that the IEDs involved provide comparable functionalities, and that an IED in a system can be replaced by another IED from the same or another manufacturer through a limited amount of reconfiguration effort. The integration effort required to make two systems cooperate depends on the specification of the interfaces: the more compatible the

interfaces are the less integration effort is required. The level of compatibility is denoted as ‘integration distance’ in Figure 4. (IEC 61850-7-1 2011: 10; Dawidczak & Englert 2010.)

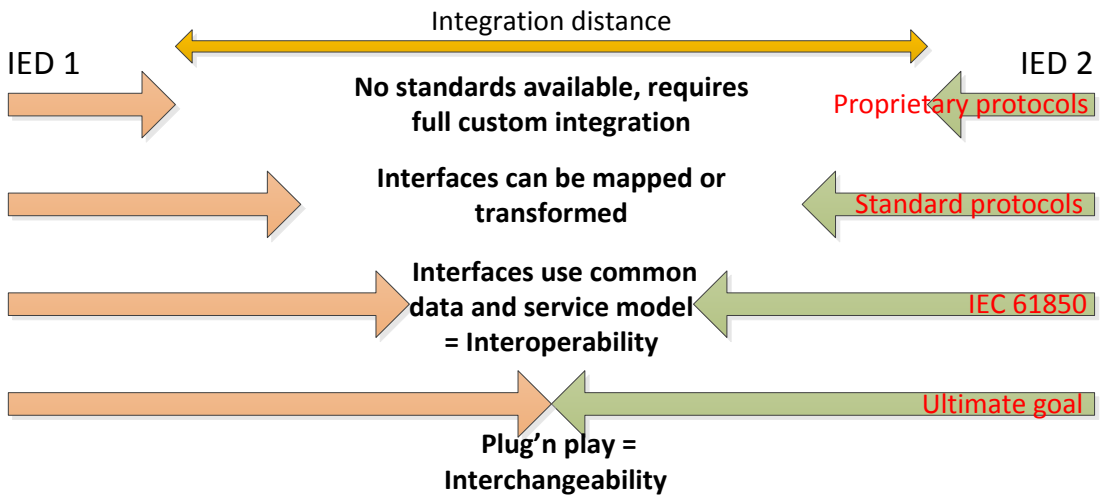


Figure 4. Integration distance between IEDs from different vendors. The longer the distance, the more engineering effort is required to make devices interoperable. (Adapted from Dawidczak & Englert 2010.)

One important thing to realize is that IEC 61850 does not cover the *interchangeability* among devices, which refers to the highest level of compatibility where an IED in a system can be replaced by another IED from another manufacturer without making any changes to the other elements in the system. To reach interchangeability, a standardization of the interfaces, functions, and algorithms is required, but as IEC 61850 clearly states, the standardization of functions is beyond its scope. Interchangeability can currently be achieved only between identical IEDs. (Dawidczak & Englert 2010; IEC 61850-1 2003: 9-10.)

2.4. Distributed functions and logical nodes

All known application functions of an SAS have been identified in IEC 61850. These functions are control and supervision, as well as protection and monitoring of the primary equipment and of the grid. The functions are composed of multiple

subfunctions, such as individual measurement and control functions implemented in different IEDs. The functions of an SAS may therefore be split into their constitutional functional parts. An IED, called *physical device* in this context, can be configured to implement one or more functions. (IEC 61850-1 2003: 12, 14; IEC 61850-7-1 2011: 16, 17.)

The *decomposition process* of IEC 61850 decomposes the application functions of an SAS into the smallest function parts which exchange data: the *logical nodes*. A logical node can be described as a virtual representation, or functional model, of a real device. For example, the logical node XCBR represents a real circuit breaker, modeled as a function. The logical nodes may be allocated to different physical devices at the same, or different levels of the functional hierarchy (see Figure 2). The allocation is not fixed and any allocation should be supported by the standard. After the allocation, the logical nodes residing in the different physical devices can exchange information over the network through information exchange services. The information exchange requires that the physical devices are interoperable. The logical nodes can take on the client/server or publisher/subscriber roles, depending on the type of application. (IEC 61850-7-1 2011: 17, 27, 77; IEC 61850-1 2003: 12, 14; Ozansoy 2006: 41-42.)

The logical node concept is illustrated in Figure 5. Each physical device (PD) contains a *logical node zero* (LN0) which refers to the information regarding the device itself. The logical nodes (LNs) are allocated to functions (F) and physical devices. The logical nodes are linked by logical connections (LC), and the physical devices are linked by physical connections (PCs). Each logical node is a part of a physical device, and each logical connection is a part of a physical connection. The *System Configuration description Language* (SCL), described in Section 2.6, is needed to control the allocation of the logical nodes to physical devices. (IEC 61850-5 2003: 20-21.)

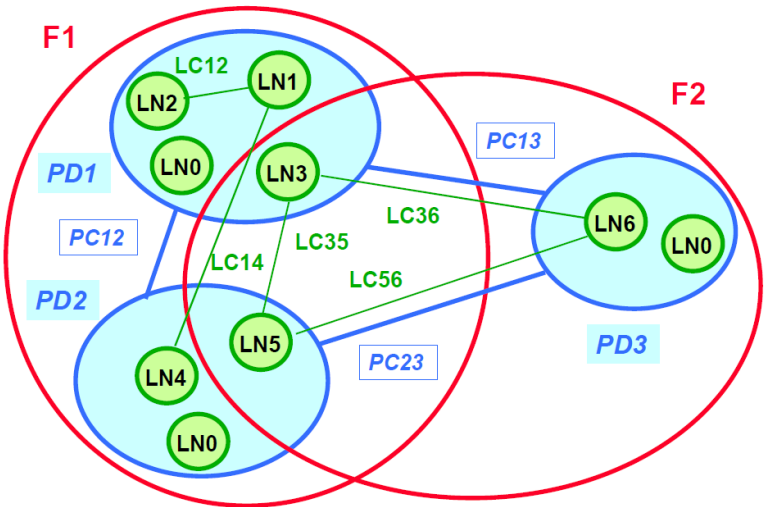


Figure 5. The logical node and link concept, where F = substation function, LN = logical node, PD = physical device, PC = physical connection, LC = logical connection. The logical nodes that make up a function can be freely distributed to different physical devices. (IEC 61850-5 2003: 21.)

The logical connections between the logical nodes can be direct, indirect, or even a combination of the two, i.e., client/server, SV, and GOOSE messaging individually or combined. The whole substation communication system is built as a *distributed* system consisting of a collection of interacting logical nodes that are logically connected by logical connections. In case of a failure of a logical node or a related link, a function might get blocked completely, or show a *graceful degradation*[6]. (Liang & Campbell 2008; IEC 61850-1 2003: 14.)

A function which is performed by two or more logical nodes that are residing in different physical devices is called a *distributed function*. The logical connection, or communication, between these logical nodes, when modelled in SCL, is called *data flow*. IEC 61850-1 states the following: "Data flow is used to understand the communication interfaces that must support the exchange of information between

[6] The ability of a computer or network to maintain limited functionality even when a large portion of it has been destroyed or rendered inoperative. (SearchNetworking 2013.)

distributed functional components and the functional performance requirements." The existence of a distributed function is thus determined by the modelled data flow. (IEC 61850-1 2003: 12, 14; Blair 2013.)

Several logical nodes, residing in the same physical device, build a *logical device*, i.e., the logical device is basically a container containing a group of logical nodes representing some related functions. A logical device always contains a logical node zero (LLN0), which represents common data of the logical device. The mode of LLN0 is used to control the mode of the logical device and all the logical nodes it consists of. For example, when the function of a logical device is disabled, all the logical nodes it consists of will be disabled as well. A logical device may also contain a *logical node physical device*[7] (LPHD), which represents common data of the physical device hosting the logical device. Logical nodes thus describe the distributed functions, the subfunctions, and the functional interfaces of an SAS. (IEC 61850-1 2003: 12, 14; IEC 61850-7-1 2011: 17, 27, 64; IEC 61850-7-2 2010: 17, 38.)

An example of an over-current protection function consists of four communicating logical nodes, as illustrated in Figure 6. The current is measured by a current transformer and the sampled analogue values are communicated by the current transformer logical node, TCTR. When logical node PIOC (instantaneous overcurrent) detects that the current grows beyond a certain limit, it signals the logical node CSWI (switch controller), which in turn activates XCBR (circuit breaker), and the circuit breaker contact opens. (Ozansoy 2006: 42; IEC 61850-7-4 2010: 34, 66, 97, 105.)

[7] LPHD shall be defined in at least one logical device. (IEC 61850-7-1 2011: 64.)

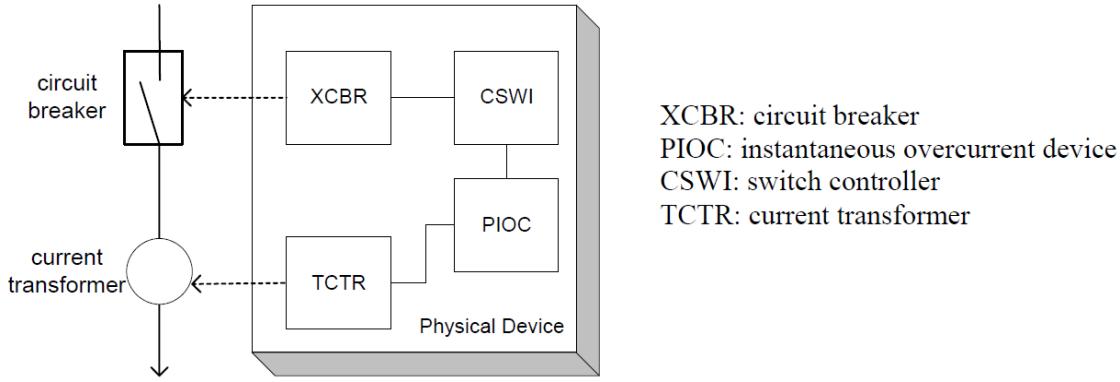


Figure 6. A simple example of an over-current protection function realized as a group of communicating logical nodes. (Ozansoy 2006: 42.)

2.5. The information class model of IEC 61850

A physical device may host zero or more *servers*[8]. A physical device may also contain one or more logical devices, which may or may not belong to a server. A logical device may contain a few or more logical nodes, i.e., an LLN0, possibly an LPHD, and one or more logical nodes representing functions. A logical device is always implemented in one physical device; therefore logical devices do not contain logical nodes from different physical devices. Each logical node contains one or more *data objects*, which represent meaningful information of applications located in a physical device. Each data object contains an application-specific set of dedicated *data attributes*, and therefore correspond to structured application data. Data attributes are the logical correspondence to physical entities, and may represent memory units, registers, communication ports, etc., presented as elementary parameter values. (Mackiewicz 2006; IEC 61850-7-1 2011: 17, 64, 76, 85; Liang & Campbell 2008; IEC 61850-7-2 2010: 38, 45; IEC 61850-6 2009: 57.)

[8] A server is basically a program running on a physical device. (Liang & Campbell 2008).

The hierarchical structure described above leads to an object oriented information class model, depicted in Figure 7, which is used to describe a real substation device. The information class model, as well as the information exchange between devices, was designed to be independent of any concrete implementation, and is therefore referred to as *abstract*. An abstract model needs to be mapped to a concrete protocol stack through a Specific Communication Service Mapping (SCSM) to become usable. The information class model, the information exchange services, and the SCSMs are discussed in more detail in Chapter 4, 'The abstract communication service interface'. (IEC 61850-7-1 2011: 17, 53; Ozansoy 2006: 37.)

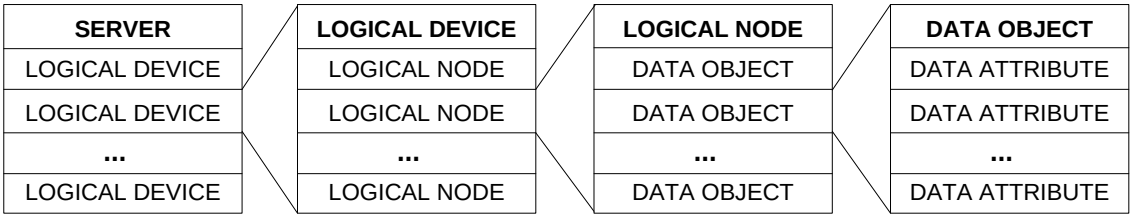


Figure 7. Hierarchy of the information class model of IEC 61850. A physical device may host zero or more servers, each of which "contains" one or more logical devices. Each logical device contains one or more logical nodes. Each logical node contains one or more data objects. Each data object contains a set of data attributes. (Adapted from Liang & Campbell 2008.)

2.6. The System Configuration description Language

The System Configuration description Language[9] (SCL) is used to describe the substation infrastructure, the SAS, and the communication between IEDs. The main purpose of the SCL is to exchange IED capability descriptions and SAS descriptions between IED engineering tools and the system engineering tool(s) from different

[9] Formerly known as the Substation Configuration Language.

vendors in a compatible way. The SCL is based on XML[10] and its semantics are defined by the IEC 61850 XML Schema. An XML Schema specifies the structure of valid XML documents. (IEC 61850-6 2009: 8; Blair 2012: 7; Goldberg 2009: 114.)

The electrical topology of the substation can be described by a single line diagram, which contains the different voltage levels, transformers, bays, busses, switchgear etc. Annex C of IEC 61850-6 defines an SCL syntax extension that can be used for displaying the power system electrical topology as a drawing. This facilitates the development of applications for visualizing the power system, the location of IEDs, and their communication services. The visualization could then be linked to real-time data from IEDs, creating a substation monitoring application. (IEC 61850-6 2009: 153; Blair, Coffele, Booth & Burt 2012; Apostolov 2010.)

2.6.1. Configuration tools and SCL file types

The *IED Configuration Tool* is a manufacturer-specific, and possibly also IED-specific, software tool that is responsible for the data model of the IED. It generates IED-specific configuration files. The *System Configuration Tool* is an IED-independent system level tool that is responsible for the communication addressing and data flow between IEDs. It generates substation-related configuration files. The data flow is modelled by a list of signals that are the input of a logical node. If several logical nodes need to access the input data, it should be mapped to the LLN0 which represents the whole logical device. A *System Specification Tool* is used to generate files that specify the substation structure by a single line diagram. (IEC 61850-6 2009: 12, 15, 26.)

There are several different subtypes of the SCL file used for the data exchange between the different configuration tools. These different file types can be distinguished by their file extensions—the extension being the name of the file type. The configuration tools

[10] eXtensible Markup Language v1.0

parse the SCL files and validate the SCL syntax and structure using IEC 61850-specific XML Schema files. Each SCL file should contain a version and revision number to distinguish different versions of the same file. This means that each tool has to keep the version and revision number information of the last file exported, or read back the last existing file to find out its version. Figure 8 depicts the *substation configuration process*[11] with the different tools and SCL file types involved. (IEC 61850-6 2009: 26-27; Yongli, Dewen, Yan & Wenqing 2009.)

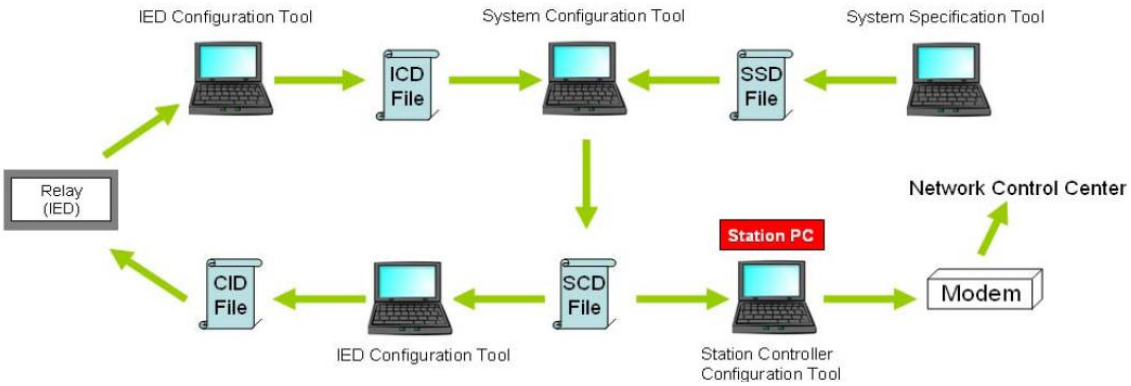


Figure 8. Conceptual substation configuration process using different software tools and different kinds of SCL files. The configuration of a substation is an iterative process where the different SCL files get exchanged between the engineering tools. (Aguilar & Ariza 2010.)

The contents of an SCL file can be conceptually divided into five different sections, each describing a distinct part of the system. The sections are called header, substation, communication system, IED, and data type templates. The header section provides information and processing instructions to the XML parser, for example information on which XML Schema shall be used when validating the file. The substation section describes the functional structure of the substation and identifies the primary devices and their electrical connections. The IED section describes the configuration of an IED and its communication services. The communication system section describes the

[11] Described in Section 2.6.2

possibilities of direct communication between logical nodes by means of subnetworks and *Service Access Points*[12] (SAPs). The data type templates define the instantiable logical node types, data object types, data attributes, and enumerate data types. (Yun 2011.)

The IED Capability Description (ICD) file is used when data is transferred from the IED Configuration Tool to the System Configuration Tool. The file must contain an IED section, which describes the functional capabilities of the IED, and a data type templates section, which describes the data types instantiated in the IED. As the IED has not been configured for the project, its name shall be TEMPLATE. The ICD file may also contain an optional substation section, where the substation name shall be TEMPLATE, and a communication section, defining possible default addresses of the IED. IEC 61850 does not specify from where the ICD file should originate. In the general case, it is stored in the IED's memory at the factory. Another possibility is that the vendor provides software tools that output the ICD file. (IEC 61850-6 2009: 26; Apostolov 2010; Yun 2011.)

The Instantiated IED Description (IID) file is also used in the data transfer from the IED Configuration Tool to the System Configuration Tool. It describes a single IED configured for the project, IED instance value changes, or data model modifications. If the IED has a project specific name it may also have project specific addresses, a data model with preconfigured *DataSet*[13] definitions, and logical nodes linked to the project specific single line diagram. IID files are used for IEDs whose number of logical node instances depends on the single line diagram or on other IEDs. (IEC 61850-6 2009: 26-27.)

The System Specification Description (SSD) file is used in the data transfer from a System Specification Tool to the System Configuration Tool. It describes the substation

[12] Abstraction of a network address by which a device is connected to a subnetwork.

[13] A list of references to data attributes the values of which shall be sent as a GOOSE or SV message.

by a single line diagram and the functional requirements represented by logical nodes. The file contains a substation section, and may contain the required data type templates and logical node definitions. The logical nodes allocated to the substation section that are not yet allocated to specific IEDs shall have the IED name reference set to 'None'. (IEC 61850-6 2009: 27.)

The System Configuration Description (SCD) file is used in the data transfer from the System Configuration Tool to the IED Configuration Tool(s). It contains the definitions of all IEDs and information regarding data flow, data types, communication configurations, and substation description. The IEDs in the SCD file have been further configured from their default state to operate within the SAS. The SCD files can be used to configure the individual IEDs. (IEC 61850-6 2009: 27; Apostolov 2010.)

The Configured IED Description (CID) file is used in the data transfer from the IED Configuration Tool to an IED. The file is a stripped down SCD file, i.e., it represents a single IED section of the SCD file, thus providing a restricted view of the source IEDs. It describes the communication related part of an instantiated IED within a project, and contains the substation specific names and addresses. (IEC 61850-6 2009: 27.)

2.6.2. The substation configuration process

In the substation configuration (engineering) process, the different SCL files are exchanged between the different configuration tools. The tools are in general not allowed to modify the files they import; instead they use the information from the imported files to generate a new type of file, as depicted in Figure 8. The IED Configuration Tool begins the configuration process by importing an ICD file from an IED, or by creating a new ICD file. The IED Configuration Tool can access the internal functions of an IED, and is needed to set up the following features (IEC 61850-6 2009: 15; Aguilar & Ariza 2010):

- Logic and trip equations
- Graphical display on an IED's HMI
- Internal mappings

- Non-IEC 61850 parameters

The System Configuration Tool imports the preconfigured ICD file which can be described as an IED template, and instantiates a project specific IED. Another alternative is to import an IID file, representing a preconfigured IED or an IED to which modifications have been made during the configuration process. The tool also obtains the substation structure from an SSD file created by the System Specification Tool. After the extraction of the required data from the ICD and SSD files, an SCD file describing the complete substation configuration, is generated. GOOSE messages can be configured by specifying the publishers and subscribers of messages. (Aguilar & Ariza 2010; IEC 61850-6 2009: 14-15; Yashwant & Swarup 2011.)

The SCD file is then imported by the IED Configuration Tool, which extracts the information needed for a specific IED from the file. The extracted information becomes the content of a CID file, i.e., the CID file describes an instantiated IED with device specific configuration data. The CID file is then uploaded to the IED. The use of a CID file to configure an IED is optional. (IEC 61850-6 2009: 15; Aguilar & Ariza 2010; Yashwant & Swarup 2011.)

If IED related data has to be changed during the configuration process, an IID file can be used to update the IED data within the system. In the next iteration of the configuration process, the IID file is imported by the System Configuration Tool. In case the IED described by the IID file does not exist in the SCD file, it can be imported and instantiated as a project specific IED without any (data flow) references to other IEDs. The required references need to be established by the System Configuration Tool, which then generates the next revision of the SCD file. If the IED already exists in the SCD file, the data model part and its values in the IID file replaces the old values in the SCD file. (IEC 61850-6 2009: 14-15.)

2.6.3. Structure of the SCL file

Examples of the different sections of the SCL file will be presented next. The examples are taken from the SCD file of the GOOSE software project that will be introduced in

Chapter 3. The project involves no substation as it merely consists of two IEDs interconnected through a switch. The examples, although somewhat incomplete, show the most important concepts of the SCL. The complete SCD file used in the GOOSE software project can be found in Appendix 2.

An example of the header section is given in Figure 9. The first line of an SCL document always contains the *XML declaration element*, enclosed in the <? and ?> characters. XML *attributes* are used to include additional information to the element. The attribute version denotes which version of XML that is being used, and the attribute encoding indicates the method in which characters are encoded. UTF-8 stands for Unicode Transformation Format (8-bit). This specification is required by XML software to able to display characters in the Unicode character set correctly on the screen. (Goldberg 2009: 4, 251-252).

```
<?xml version="1.0" encoding="utf-8"?>
<SCL xmlns="http://www.iec.ch/61850/2006/SCL" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.iec.ch/61850/2006/SCL">
  <Header id="SCL Example T1-1" nameStructure="IEDName" />
```

Figure 9. An example of the header section of an SCL file. The default namespace, xmlns, is IEC 61850-specific.

The second line in Figure 9 introduces the *root element*, which is the main XML element that contains all other elements. The name of the root element is SCL, indicating that the file is an SCL document. XML elements are always enclosed inside angle brackets. The attribute xmlns, which is an abbreviation for XML *namespace*, indicates the default namespace for the SCL document. A namespace is a collection of related elements and attributes, identified by a namespace name. Names that are part of one namespace do not interfere with the same names that are part of another namespace. In other words, two elements can be assigned the same name if they belong to different namespaces. The namespace name, i.e., the value of attribute xmlns, must be unique,

permanent, and written in the form of a URI[14], typically in URL[15] format. The URL does not need to point to an actual file—URLs are used because they are unique. XML software does not even try to locate whatever the URL is pointing to. (Goldberg 2009: 4, 116, 164-165, 169.)

The name of the namespace is used by the XML parser to associate the SCL document with a specific XML Schema. The default namespace name for an SCL document is http://www.iec.ch/61850/2006/SCL which is the namespace specified by IEC containing all the XML elements[16], attributes, and data types that can be used in the creation of a valid SCL file. The semantics and structure of the SCL file will thus be validated against the IEC 61850 XML Schema defined in IEC 61850-6. (Goldberg 2009: 116, 170.)

Prefixes are used as shorthand for the namespace names. The combination xmlns:xsi in the root element declares the prefix xsi which refers to the unique URL http://www.w3.org/2001/XMLSchema-instance. The prefix is used to label individual elements, i.e., to assign elements to a particular namespace. xsi would thus be used to label any valid XML Schema element not belonging to the IEC 61850-specific XML Schema. Elements without a prefix belong to the default namespace. (Goldberg 2009: 165-166, 169.)

An example of the communication section of an SCL file is given in Figure 10. The communication system consists of a subnetwork named W01. W01 is of type 8-MMS which refers to part 8-1 of the standard, i.e., the station bus. The given bitrate of 10 b/s is apparently incorrect—the correct XML element for a Fast Ethernet network is <BitRate multiplier="M" unit="b/s">100</BitRate>. A physical device named

[14] Uniform Resource Identifier

[15] Uniform Resource Locator

[16] Element definitions and descriptions of element attributes can be found in IEC 61850-6.

Laptop is connected to the subnetwork via a SAP named S1. A GSE (GOOSE) control block named Status, contained in logical device C1 of Laptop, needs to be assigned (IEC 61850-8-1 2011: 143; IEC 61850-6 2009: 81):

- a multicast address from the address range specified for GOOSE in Annex B of IEC 61850-8-1,
- an APPID (application identifier) which is a system wide unique identification of the application to which the GOOSE message belongs, and
- a VLAN priority value of 4, as is commonly used for GOOSE and SV messages.

```
<Communication>
  <SubNetwork name="W01" type="8-MMS">
    <Text>Station bus</Text>
    <BitRate unit="b/s">10</BitRate>
    <ConnectedAP iedName="Laptop" apName="S1">
      <Address>
        <P type="IP">10.4.128.104</P>
        <P type="IP-SUBNET">255.255.255.0</P>
        <P type="IP-GATEWAY">10.4.128.253</P>
        <P type="OSI-TSEL">00000001</P>
        <P type="OSI-PSEL">01</P>
        <P type="OSI-SSEL">01</P>
      </Address>
      <GSE ldInst="C1" cbName="Status">
        <Address>
          <P type="MAC-Address">01-0C-CD-01-00-05</P>
          <P type="APPID">3002</P>
          <P type="VLAN-PRIORITY">4</P>
        </Address>
      </GSE>
      </ConnectedAP>
  </SubNetwork>
</Communication>
```

Figure 10. An example of the communication section of an SCL file. The communication system consists of a subnetwork of station bus type. For brevity, only one connected IED is shown. The IP address of the IED is needed as well as the multicast address, application ID, and VLAN priority used by the GSE control block contained in the IED.

An example of the IED section is given in Figure 11, where the hierarchies of the IEDs involved are clearly distinguishable from the SCL code. IED Laptop is connected to a

subnetwork through a SAP named S1.[17] The SAP has a server containing the mandatory element Authentication, and a logical device named C1. Authentication defines, in case of a device description, the authentication possibilities, and in the case of an instantiated device, the methods to be used for authentication. (IEC 61850-6 2009: 68.) C1 contains two logical nodes, the LN0 and the DCIPa, which are instances of the logical node classes LLN0 and DCIP, respectively. LN0 has a DataSet named Status which references data attribute stVal, which resides in the data object EngOnOff of logical node DCIPa. A DataSet can contain references to data objects or data attributes in any logical node on the same server where the DataSet itself is defined. (IEC 61850-7-2 2010: 63.) The DataSet Status is governed by the GSE control block Status.[18] The logical node DCIPa has two inputs, each referencing a data attribute called stVal. These are two separate data attributes residing in two separate logical nodes in IED VAMP.

IED VAMP mainly follows the same structure, except that the logical node LLN0_0 contains both a DataSet and an external reference. The DataSet DSG1 references the two data attributes called stVal residing in the logical nodes VI1GGIO137 and VI2GGIO138. The names of these two logical nodes can be derived from the concatenation of the prefixes VI1 and VI2, the logical node class GGIO, and the suffixes 137 and 138. For brevity, they were left out from the example. The input references data attribute stVal of IED Laptop. The same data attribute is also referenced by DataSet Status of IED Laptop. An input in one device can "externally reference" only a data object/attribute which is referenced by (included in) a DataSet of the "external" device. It is very important that these cross-references (dataflow) between DataSets in one device and inputs in another device are configured correctly.

[17] No services were defined; the names of any services should be listed between the IED name and the SAP name. Consult IEC 61850-6 for examples.

[18] A DataSet and the control block that governs it do not need to have the same name.

```
<IED name="Laptop">
  <AccessPoint name="S1">
    <Server>
      <Authentication none="true"/>
      <LDevice inst="C1">
        <LN0 lnType="LN0" lnClass="LLN0" inst="">
          <DataSet name="Status">
            <FCDA ldInst="C1" prefix="" lnInst="1" lnClass="DCIP"
            doName="EngOnOff" daName="stVal" fc="ST" />
          </DataSet>
          <GSEControl name="Status" datSet="Status" appID="Status"
          confRev="1"/>
        </LN0>
        <LN lnType="DCIPa" lnClass="DCIP" inst="1" >
          <Inputs>
            <ExtRef iedName="VAMP" ldInst="Relay" prefix="VI1" lnClass="GGIO"
            lnInst="137" doName="SPCSO" daName="stVal" />
            <ExtRef iedName="VAMP" ldInst="Relay" prefix="VI2" lnClass="GGIO"
            lnInst="138" doName="SPCSO" daName="stVal" />
          </Inputs>
        </LN>
      </LDevice>
    </Server>
  </AccessPoint>
</IED>
<IED name="VAMP" type="VAMP 50">
 <AccessPoint name="P1">
   <Server>
    <Authentication none="true"/>
    <LDevice inst="Relay">
      <LN0 lnType="LLN0_0" lnClass="LLN0" inst="">
        <DataSet name="DSG1">
          <FCDA ldInst="Relay" prefix="VI1" lnClass="GGIO" lnInst="137"
          doName="SPCSO" daName="stVal" fc="ST"/>
          <FCDA ldInst="Relay" prefix="VI2" lnClass="GGIO" lnInst="138"
          doName="SPCSO" daName="stVal" fc="ST"/>
        </DataSet>
        <Inputs>
          <ExtRef intAddr="NI1" iedName="Laptop" ldInst="C1" prefix=""
          lnClass="DCIP" lnInst="1" doName="EngOnOff" daName="stVal"/>
        </Inputs>
        <GSEControl name="gcb1" type="GOOSE" appID="VAMP" confRev="1"
        datSet="DSG1"/>
      </LN0>
    </LDevice>
   </Server>
  </AccessPoint>
</IED>
```

Figure 11. An example of the IED section of an SCL file. Notice the use of references between DataSets in one IED and inputs (ExtRef) in the other IED.

An example of the data type templates section of an SCL file is given in Figure 12. The data type templates section lists the instantiable logical node types, data object types, data attributes, and enumerated types. The logical node DCIPa is instantiated from the *compatible logical node class* DCIP, and it contains a data object named EngOnOff. EngOnOff is of type (is derived from) simpleSPS, which is merely a subset of the data attributes contained in the *common data class* SPS—simpleSPS implements only the mandatory data attributes stVal (a Boolean value), q (quality), and t (timestamp), required for instantiation of the SPS class. These data attributes are functionally constrained by the *functional constraint* value ST (status). The attributes dchg (data change) and qchg (quality change) are the *trigger options*, i.e., the reasons that trigger the automatic sending of a GOOSE message containing the values of the data attributes that are referenced by the DataSet belonging to the GSE (GOOSE) control block. More details will be given in Chapter 4, 'The Abstract Communication Service Interface'.

```
<DataTypeTemplates>
  <LNodeType id="DCIPa" lnClass="DCIP">
    <DO name="EngOnOff" type="simpleSPS"/>
  </LNodeType>
  <DOType id="simpleSPS" cdc="SPS">
    <DA name="stVal" fc="ST" bType="BOOLEAN" dchg="true"/>
    <DA name="q" fc="ST" bType="Quality" qchg="true"/>
    <DA name="t" fc="ST" bType="Timestamp"/>
  </DOType>
  <DAType id="ScaledValueConfig">
    <BDA name="scaleFactor" bType="FLOAT32"/>
    <BDA name="offset" bType="FLOAT32"/>
  </DAType>
  <EnumType id="Health">
    <EnumVal ord="1">Ok</EnumVal>
    <EnumVal ord="2">Warning</EnumVal>
    <EnumVal ord="3">Alarm</EnumVal>
  </EnumType>
</DataTypeTemplates>
```

Figure 12. An example of the data type templates section of an SCL file. It contains templates for the instantiable logical node types, data object types, data attributes, and enumerated data types.

3. THE GOOSE SOFTWARE APPLICATION

The goal for the practical part of this thesis was to realize interoperable GOOSE communication between two different IEDs. A simple application based on an open source platform was developed in order to enable communication between a Wärtsilä Local Display Unit 20 (LDU) and a Vamp 50 overcurrent and earthfault protection relay. A small Ethernet LAN was established by interconnecting the LDU and Vamp relay through a switch, which was configured to strip the VLAN tags from the Ethernet packets. The network traffic was monitored with Wireshark and PuTTY that were running on a laptop PC. PuTTY was also used to start up the GOOSE application in the LDU. The hardware setup is illustrated in Figure 13.

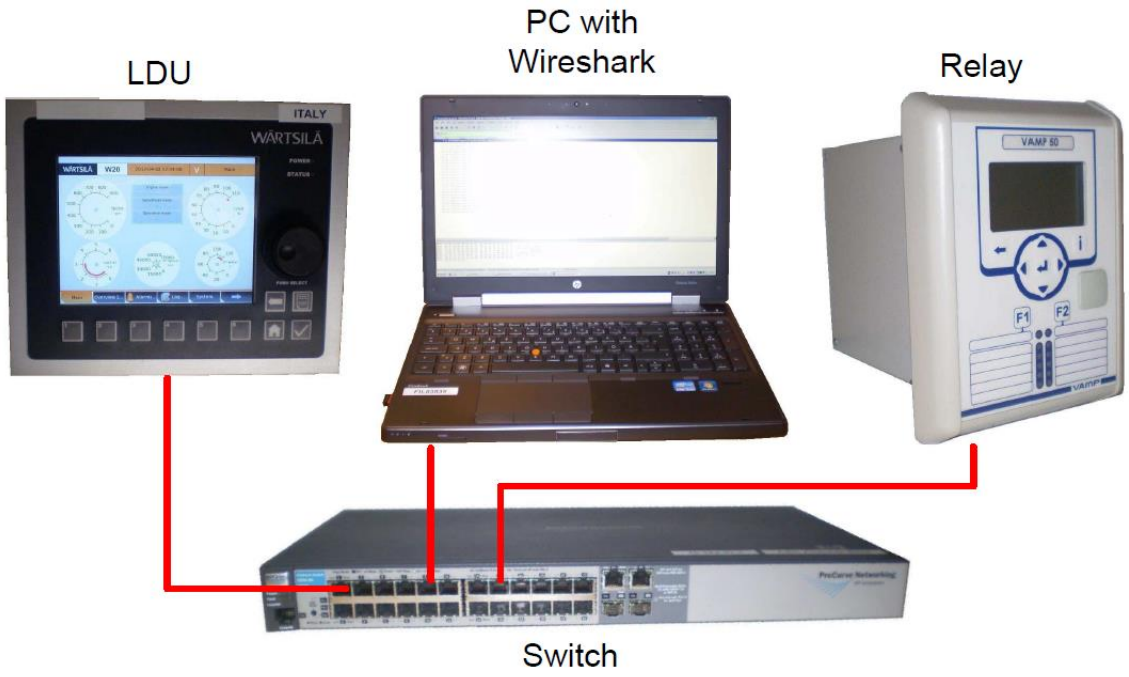


Figure 13. The hardware setup. The LDU and Vamp IEDs communicate over an Ethernet network through a switch and the network traffic is monitored by Wireshark running on a laptop PC.

The LDU is based on the PowerPC processor architecture, and has a particular distribution of the Linux operating system running on it. The open source Oracle VM Virtualbox platform is capable of simulating the Linux development environment used

for creating software for the LDU. The development environment was used on the laptop to cross-compile code for the LDU. The following sections present the software used in the development of the GOOSE messaging application. Chapter 3 is concluded by a brief discussion of the results.

3.1. The Rapid61850 platform

Rapid61850 is a software platform developed at the University of Strathclyde, Glasgow, UK. It is stated to be the first open source implementation of IEC 61850, and its main intended application is to implement IEDs for rapid prototyping of protection, control, and automation systems in both research and education. The idea is that the system designer can focus on the design and implementation of the protection scheme, rather than on the underlying communications infrastructure. The platform, which is based on the Eclipse Modelling Framework (EMF), takes a System Configuration Description (SCD) file as an input, and automatically generates the low-level communications code required for implementing GOOSE and Sampled Values messaging in the C programming language. The generated communication stack is hardcoded and therefore very efficient at run-time—the IED does not need to interpret the SCD file at run-time or maintain an internal model of the SCD. (Blair 2012; Blair, Booth & Burt 2011a; Blair, Coffele, Booth & Burt 2012).

The code generation process is depicted in Figure 14. The EMF automatically generates a Java model of the IEC 61850 XML Schema. It also generates an XML parser, tailored to the Schema, for parsing SCD files. An SCD file is then imported to the EMF platform, and the XML parser performs syntactical and semantic validation of the SCD file. The validation process is separate from the code generation process and extensively uses the EMF Model Query framework for searching and filtering SCD data. After successful validation, the parser generates an instance of a hierarchical model containing Java objects. The objects are automatically populated with data from the SCD file, and the model can be further manipulated in software. Java Emitter Templates (JET) files are then used to transform the Java model instance into a C implementation.

The JET files define the generic structure of the C source and header files. The generated C code implements all IEDs specified in the SCD file, and may be used as part of a C/C++ program. An example SCD file and a main.c file are also provided by the software package. (Blair 2012; Blair Booth & Burt 2011a; Blair, Coffele, Booth & Burt 2012).

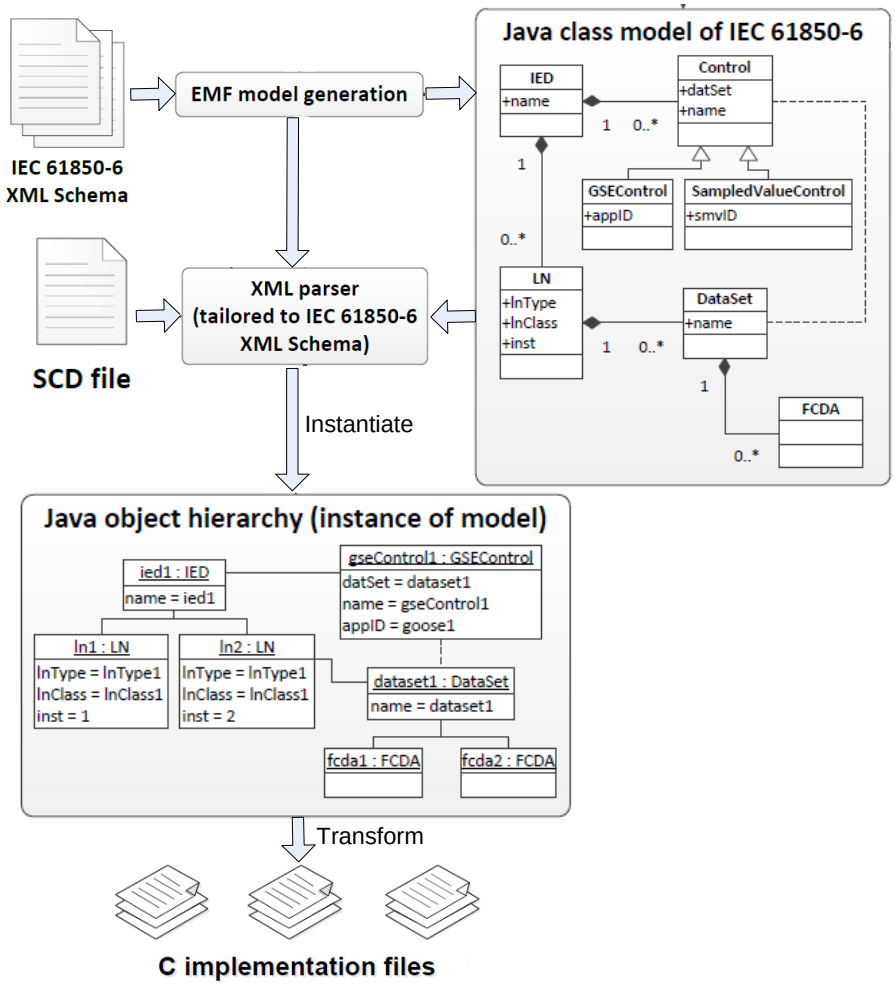


Figure 14. The code generation process. The EMF platform generates a Java model and an XML parser from the IEC 61850 XML Schema files, validates the SCD file used as an input, generates an instance of the Java model according to the structure in the SCD file, populates the instance with data from the SCD file, and finally transforms the Java objects into C code. (Adapted from Blair, Coffele, Booth & Burt 2012; Blair, Booth & Burt 2011b.)

Each data type in the SCD file is mapped to a C data structure, resulting in a hierarchy of C data structures. Primitive types, such as integer and floating-point numbers, are mapped to generic primitive types, which are then mapped to device-specific C primitive types of appropriate byte-length and sign. An example of the mapping of the common data class SAV (Sampled Value) from SCL to C is given in Figure 15. (Blair, Coffele, Booth & Burt 2012.)

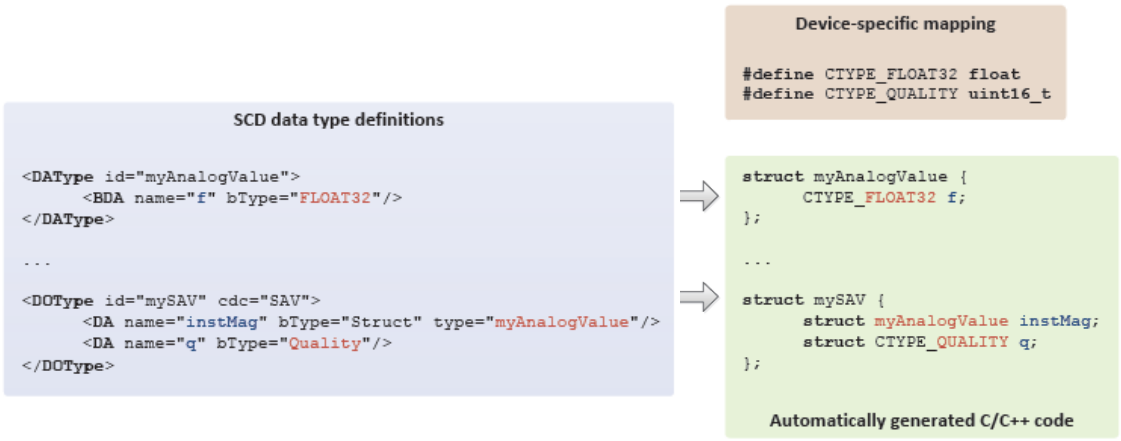


Figure 15. The mapping of hierarchical SCL data types creates C data structures with the same hierarchy in the resulting code. (Adapted from Blair, Coffele, Booth & Burt 2012.)

The software is intended to be platform-independent and lightweight enough to run on embedded devices, and therefore only GOOSE and SV messaging are implemented. The architecture may be extended in the future to support other features of IEC 61850, like the MMS services. (Blair, Booth & Burt 2011a.)

The configuration process required to set up the Rapid61850 platform is rather complicated. The software setup was done by following the instructions in the README.md file found at (Blair 2012).

3.2. The Vampset IED Configuration Tool

Vampset is an IED Configuration Tool provided by Vamp Ltd. for configuring Vamp IEDs. In order to configure an IED in Vampset, the IED must first be connected to the PC through an Ethernet network or USB. After that the connection is established, the ICD file residing in the IED is automatically downloaded to Vampset—this procedure is often referred to as "self-description" of a device in IEC 61850.

The IEC 61850 server interface of Vamp supports (VAMP 2009: 1):

- a configurable data model—selection of logical nodes corresponding to active application functions,
- configurable pre-defined DataSets,
- dynamic DataSets created by clients,
- reporting functions with buffered and unbuffered report control blocks,
- a control model; direct with normal security,
- configurable GOOSE publisher DataSets, and
- configurable filters for GOOSE subscriber inputs.

A Vamp IED receiving GOOSE messages can map the binary values of the message to its 64 input points, i.e., network inputs NI1 to NI64. Vamp IEDs can transmit GOOSE messages which consist of a maximum of 8 Boolean data attribute values. Two GOOSE control blocks (GCB1 and GCB2) are available for controlling the transmission of GOOSE messages—the maximum number of data points in one device is therefore 16. (VAMP 2009: 5.)

The following figures reflect the configurations done in Vampset to enable the Vamp IED to transmit and receive GOOSE messages. The configuration process was initiated by resetting all the current settings—all report control blocks were disabled, whereas the GCB1 and GOOSE subscription were enabled. The GOOSE configuration view in Vampset is illustrated in Figure 16. Different Application IDs must be used for GCB1 and GCB2, and for every IED communicating with the device. Vamp IEDs can receive GOOSE messages only via one MAC address, therefore all devices transmitting data

to the Vamp IED must have the same MAC address set in their GOOSE control blocks. The different sources are distinguished by their unique Application IDs. (VAMP 2010a: 6; VAMP 2010b: 3.)

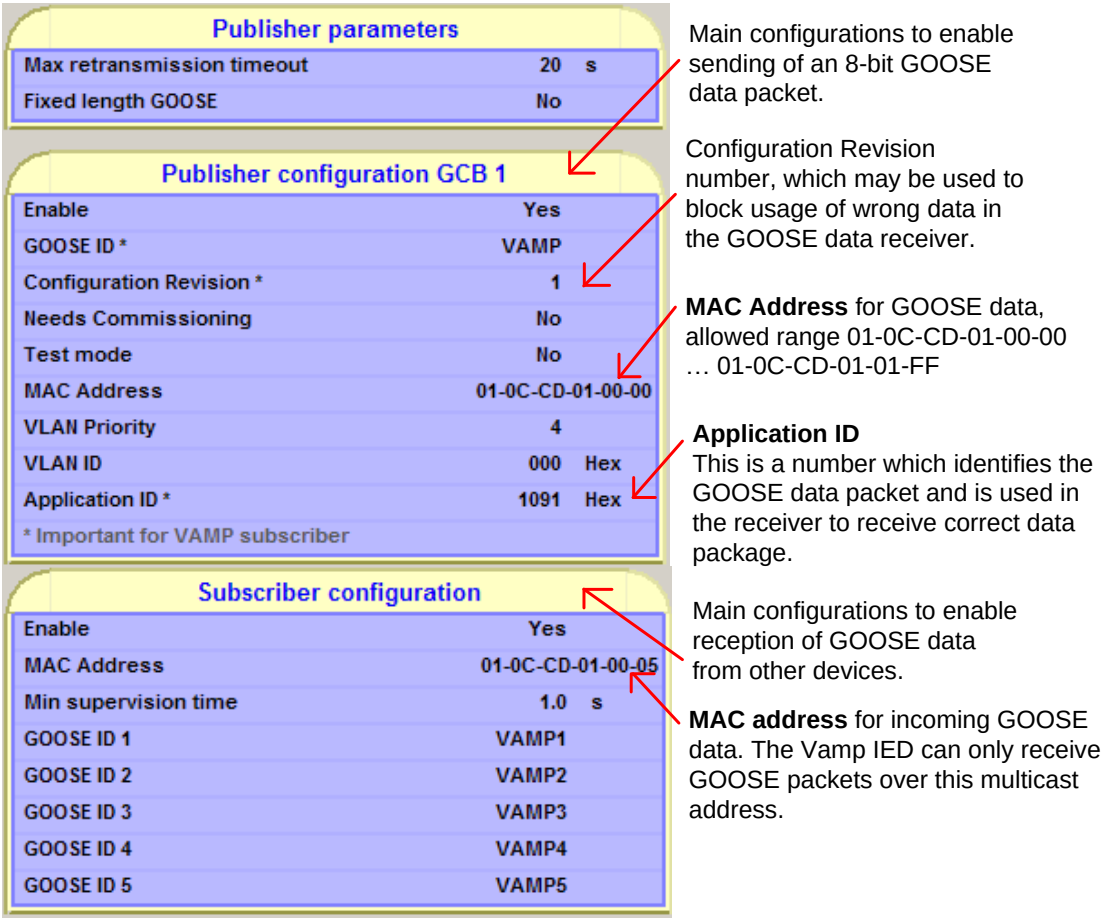


Figure 16. The GOOSE configuration view in Vampset shows the main parameters for configuring the GOOSE publisher and subscriber functions. (Screenshot from Vampset.)

The logical nodes (functions) which are to be used via the IEC 61850 interface are selected in the IEC 61850 data map view, as illustrated in Figure 17. Virtual inputs 1 and 2, which correspond to the F1 and F2 buttons on the front panel of Vamp 50, were set in use. The virtual inputs 1 and 2 are data attributes residing in the logical nodes VI1GGIO137 and VI2GGIO138, respectively. The user can also select which logical nodes shall relate to the three available report control block DataSets. Some clients

may create their own so called dynamic data sets and assign these to report control blocks. Both *persistent* and *non-persistent* DataSets are supported. The meaning of "persistent" and "non-persistent" is explained in Section 4.7, 'The DataSet class'. (VAMP 2009: 3-4.)

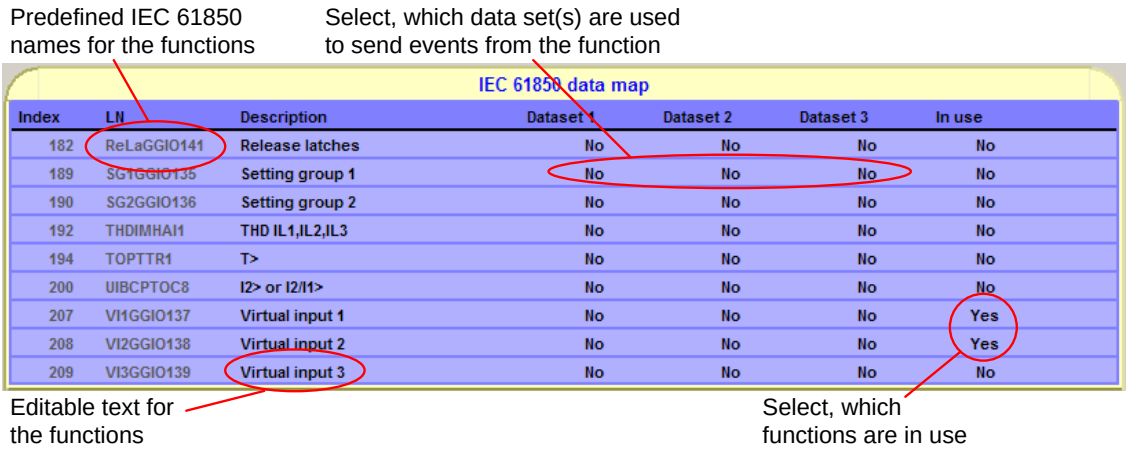


Figure 17. The IEC 61850 data map view shows the logical nodes (functions) which are used and whose data attributes can be included in DataSets for reporting functions. (Screenshot from Vampset.)

The data attributes of the logical nodes set in use in the IEC 61850 data map view can now be included in a GOOSE DataSet, governed by GCB1 or GCB2, as illustrated in Figure 18.

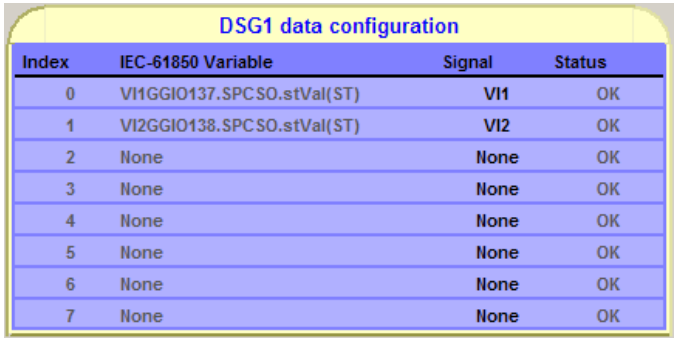
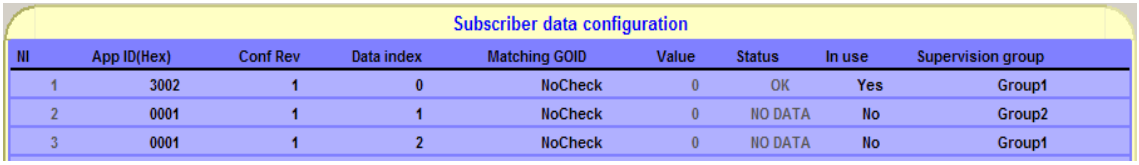


Figure 18. The GOOSE DataSet. The user can select up to eight Boolean data attributes to be sent as a GOOSE message. (Screenshot from Vampset.)

The GOOSE messages that the IED shall subscribe to are selected in the Subscriber data configuration window, illustrated in Figure 19. The App ID of the IED to receive GOOSE messages from must be set, along with the data index of the desired data attribute in the GOOSE packet. The application being developed needed to receive only one single data attribute.



Subscriber data configuration

NI	App ID(Hex)	Conf Rev	Data index	Matching GOID	Value	Status	In use	Supervision group
1	3002	1	0	NoCheck	0	OK	Yes	Group1
2	0001	1	1	NoCheck	0	NO DATA	No	Group2
3	0001	1	2	NoCheck	0	NO DATA	No	Group1

Figure 19. The GOOSE subscriber data configuration view. The IED is configured to receive a GOOSE message with App ID 3002, and to map the data attribute at index 0 in the message internally to Network Input NI1. (Screenshot from Vampset.)

After that any configurations have been done in Vampset the IED must be updated by uploading the new configurations to it. Vampset can export the ICD file, which can subsequently be opened in a System Configuration Tool or in an XML editor. More information on Vampset can be found in the documents at Vamp's website[19].

3.3. Development of the SCD file

The system configuration began by taking the SCD file scd.xml provided by Rapid61850 and stripping it of everything unnecessary: everything but the header section, an IED section, and the communications and data type templates sections belonging to that IED was removed. Then, by using the IED and its communication and data type templates as a framework, a new IED named Laptop was created. No System Configuration Tool was used in this project; the SCD file was edited manually in an

[19] http://www2.schneider-electric.com/sites/corporate/en/products-services/former-brands/vamp/vamp.page

XML editor. Section 3.3.1 explains how IED Laptop was derived. Section 3.3.2 explains how IED VAMP, generated by Vampset, was added to the SCD file.

3.3.1. IED Laptop

IED Laptop was assigned the logical node DCIP, which represents reciprocating engine characteristics, measured values, and controls, as outlined by IEC 61850-7-420 and given in Table 1. The first letter (D) in the logical node name is the group indicator, indicating that DCIP belongs to a group of logical nodes defined for the DER domain. Only the mandatory (M) data object in the DCIP class, EngOnOff, was used. EngOnOff can acquire the Boolean values of True or False, and is derived from the common data class SPS. Common data classes (CDCs) are explained in more detail in Section 4.6, 'Data object classes'. (IEC 61850-7-420 2009: 56; IEC 61850-7-1 2011: 18.)

Table 1. Portion of the class definition for the logical node class DCIP. The data object EngOnOff, derived from CDC SPS, can acquire the Boolean values True or False, and is mandatory in an instance of the DCIP class. (IEC 61850-7-420 2009: 57.)

<table>
<tr><th colspan="5">DCIP class</th></tr>
<tr><th>Data object name</th><th>Common data class</th><th>Explanation</th><th>T</th><th>M/O/C</th></tr>
<tr><td>LNName</td><td></td><td>Shall be inherited from logical-node class (see IEC 61850-7-2)</td><td></td><td></td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">System logical node data</td></tr>
<tr><td></td><td></td><td>LN shall inherit all mandatory data from common logical node class</td><td></td><td>M</td></tr>
<tr><td></td><td></td><td>Data from LLN0 may optionally be used</td><td></td><td>O</td></tr>
<tr><td colspan="5">Status information</td></tr>
<tr><td>EngOnOff</td><td>SPS</td><td>Engine status:<table><tr><th>Value</th><th>Explanation</th></tr><tr><td>True</td><td>On</td></tr><tr><td>False</td><td>Off</td></tr></table></td><td></td><td>M</td></tr>
</table>

SPS stands for Single Point Status and can be found in IEC 61850-7-3. The SPS class is given in Table 2. Three data attributes are mandatory in an instance of the SPS class: stVal, which can acquire Boolean values, q (quality), and t (timestamp). stVal was used, i.e., referenced by a DataSet. stVal has the trigger option (TrgOp) dchg (data

change), which means that when stVal has a data value change from false-to-true or true-to-false, the value of stVal is sent as a GOOSE message to another IED that references stVal, i.e., IED VAMP. All three data attributes have a functional constraint (FC) meaning that they are functionally constrained data attributes (FCDAs).

Table 2. Portion of the class definition for the CDC Single Point Status (SPS). Three data attributes are mandatory when a data object is instantiated from the SPS class: stVal, q, and t. (IEC 61850-7-3 2010: 26.)

SPS class					
Data attribute name	**Type**	**FC**	**TrgOp**	**Value/Value range**	**M/O/C**
DataName	Inherited from GenDataObject Class or from GenSubDataObject Class (see IEC 61850-7-2)				
DataAttribute					
status					
stVal	BOOLEAN	ST	dchg	TRUE | FALSE	M
q	Quality	ST	qchg		M
t	TimeStamp	ST			M

The functional constraint indicates the services that are allowed to be operated on a specific data attribute: FC = ST (status information) indicates that the status information values may be read, substituted, reported or logged, but not written (by services), as given in Table 3. The data attributes of a CDC are also grouped by FCs into categories. The functional constraints can be found in IEC 61850-7-2. (Mackiewicz 2006.)

Table 3. The definition of the FC value ST. (IEC 61850-7-2 2010: 54.)

FunctionalConstraint values			
FC	**Semantic**	**Services allowed**	**Initial values/storage/ explanation**
ST	Status information	**DataAttribute** shall represent status information whose value may be read, substituted, reported, and logged but shall not be writeable.	Initial value of the **DataAttribute** shall be taken from the process.

The data type templates derived for DCIP, EngOnOff, and simpleSPS were given in the end of Section 2.6.3, ‘Structure of the SCL file’.

3.3.2. IED VAMP

The ICD file, describing the complete configuration of Vamp 50, was opened in an XML editor. A reference to the data attribute stVal of DataSet Status in IED Laptop was added, after which the updated ICD file was re-opened in Vampset and used to update Vamp 50. The information in the ICD file was then copied over to the SCD file. The addition of the VAMP IED to the SCD file demanded significant modifications to the SCL code due to the compatibility issues discovered when Rapid61850 parsed the file. The main problem was caused by the fact that Rapid61850 uses a newer namespace (2006) while Vamp uses an older one (2003). To solve the problem, the old namespaces were removed, as were incompatible parts of the data type templates section. Unused report control blocks and GOOSE network inputs were removed for convenience. The resulting SCD file that was finally accepted by the Rapid61850 parser no longer entirely conformed to the IEC 61850-6 standard due to the removed information. The modified SCD file was also not used to update the Vamp IED. However, as the SCD file was only needed to generate the communications code for the LDU, the modifications had no impact on the result—what was most important was that the MAC addresses, Application IDs, DataSet references, and input references were configured correctly.

3.4. The C source code

The finished SCD file was used as an input to the Rapid61850 parser, which then generated some C source and header files. The provided main.c file and the example functions it contained were used as a framework when new functionalities were developed. Code for reading and writing Ethernet packets is however not provided by the software platform. The code had to be included manually, as will be explained next.

3.4.1. Communication

The README.md file recommends the pcap (packet capture) API (Application Programming Interface) for handling the communication. Packet capture refers to the

action of collecting data as it travels over a network. Linux implements pcap in the libpcap library, which can be used by programs to read and write data packets directly at the DataLink layer, independently of the actual DataLink access provided by the operating system. This is consistent with the mapping of GOOSE messages directly to Ethernet frames, bypassing the middle layers. (Garcia 2008; Wikipedia 2013; Stevens 1998: 703, 708, 725.)

The libpcap library was not readily available in the LDU, and therefore a *packet socket* was created instead. In the general case, sockets represent interfaces from the upper three layers into the Transport layer. The use of sockets enables communication between applications on the same host or between applications on different hosts connected via a network (i.e., communication on two different *ranges*). A socket always exists within a communication domain, which determines the range of communication and the address format used to identify the socket. The most common socket types are stream sockets (SOCK_STREAM) and datagram sockets (SOCK_DGRAM), used with the Transport layer protocols TCP and UDP[20], respectively. A socket of type raw (SOCK_RAW) can be used to bypass the Transport layer and use the Network layer directly. (Stevens 1998: 18; Kerrisk 2010; 1150, 1162.)

The function int socket(int domain, int type, int protocol); creates a socket and returns a file descriptor used to refer to the newly created socket in later system calls. The domain argument specifies the domain for the socket, and the type argument specifies the socket type. The value of the protocol argument is in general 0. The address information of the socket is stored in a struct of type sockaddr_ll. The function call packet_socket = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL)); creates a packet socket. Packet sockets are used to send or receive packets at the DataLink layer, and can thus be used for GOOSE messaging. The code for the packet socket can be found in Appendix 3. (Kerrisk 2010: 1153; Ubuntu Manuals 2013.)

[20] User Datagram Protocol

The most important functions provided by Rapid61850 are the encoding and decoding functions for GOOSE and Sampled Values. GOOSE packets are generated by calling the function send(buffer, statusChange, timeAllowedToLive); where buffer is a pointer to a reserved memory space where the bytes are to be stored, statusChange should be 1 if any value in the DataSet has changed, and timeAllowedToLive is the maximum time in milliseconds for the receiver to wait for the next retransmission. If this time is exceeded before a new message arrives, it is assumed that a communication failure has occurred. The function returns the length of the GOOSE packet. (Blair 2012; IEC 61850-8-1 2011: 93.)

The transmission of GOOSE messages is controlled by a GOOSE Control Block. However, no trigger options are supported at present in Rapid61850. The most common trigger option is such that when the value of a data attribute referenced by a GOOSE DataSet changes, i.e., when an *internal event* has occurred, the message is transmitted. Therefore it is up to the user to create an *event monitor* that monitors the referenced data attributes. When a value changes, the send function is called and the statusChange parameter is set to 1. Upon retransmission, the value is set to 0. (Blair 2012; Blair 2013.)

The function developed for retransmitting "old" GOOSE messages is given in Figure 20. Transmission of a "new" message is done through a callback function as will be explained in Section 3.4.3. More details on Linux API functions can be found in (Stevens 1998) or the Ubuntu Manuals online.

```
len = Laptop.S1.C1.LN0.Status.send(buf, 0,
milliintervals[intervalselector]);
strncpy(socket_address.sll_addr,
Laptop.S1.C1.LN0.Status.ethHeaderData.destMACAddress, ETH_ALEN);
send_result = sendto(s, buf, len, 0,
(struct sockaddr*)&socket_address, sizeof(socket_address));
if (send_result == -1) {
    fprintf(stderr, "Send failed");
}
```

Figure 20. The functions for the encoding and sending of a GOOSE data packet. The encoding function is provided by Rapid61850 and the send function uses a packet socket.

The function for receiving a GOOSE packet is given in Figure 21. As long as there is data in the reception buffer, the packets are filtered by the multicast address range specified for GOOSE and Sampled Values in Annex B of IEC 61850-8-1.

```
do {
    len = recvfrom(s, buf, BUFFER_LENGTH, 0, NULL, NULL);
    if(len){
        gse_sv_packet_filter(buf, len);
    }
} while(len >= 0);
```

Figure 21. The functions used for receiving and filtering GOOSE messages.

A framework for an optional callback function is provided. After a received GOOSE or Sampled Values message has been successfully decoded, the callback function is called. The actual functionality of the callback function is user defined. It can, for example, be used to save the received GOOSE data to a separate memory buffer—by default, only one packet of data is saved in the buffer for each GSE Control, and that packet is overwritten when a new packet arrives. (Blair 2012.)

3.4.2. The GOOSE retransmission scheme

The GOOSE retransmission scheme, specified in IEC 61850-8-1, is not available in Rapid61850 at present. Therefore, it had to be developed and added manually. The developed algorithm is experimental and was used for demonstration purposes only. It fulfils neither the functional requirements nor the performance requirements imposed by the GOOSE protocol. A real implementation that is utilizing GOOSE trip messages would also require a real-time operating system running on the target processor.

The retransmission algorithm is given in Figure 22. At the first loop of the infinite for loop, the value of ticks is 0 and the value of intervalselector is also 0, which means that the if statement is evaluated as true. As the value of intervalselector is less than 4, it is incremented by one. Next, a GOOSE packet containing the values of the data attributes referenced by DataSet Status is generated. The packet is assigned the timeAllowedToLive value specified in the array milliintervals at the position determined by the value of intervalselector, i.e., 1024 milliseconds. The GOOSE

packet is then sent over the network using the specified multicast address. To indicate that a package was sent, a message containing the timeAllowedToLive value is printed to console. The variable ticks is reset to 0 and the program sleeps for 500 milliseconds.

The idea with the sleeping is to create the delays corresponding to the retransmission intervals of the GOOSE retransmission scheme, which is explained in more detail in Section 4.8. One loop corresponds to (approximately) a 0.5 second delay, four loops correspond to 4*500 ms = 2 seconds, 20 loops correspond to 10 seconds, and 20 loops correspond to the *keep-alive*[21] interval of 20 seconds.

Although the value of ticks was reset to 0, it is immediately incremented to 1 in the second loop by the update expression in the for loop. As the if statement is still evaluated as true, another message is transmitted. Variable ticks is again reset to 0 and the application sleeps for another 500 milliseconds.

In the third loop, the if statement is no longer evaluated as true. Therefore no message is transmitted, Intervalselector is not incremented, and ticks is not reset to 0. The if statement is evaluated as false also in the fourth and fifth loops, and the application has thus "slept" for an interval of approximately 2 seconds. In the sixth loop, the value of ticks has grown to 4, meaning that the if statement is again evaluated as true, and another message is sent.

The algorithm continues in this manner to increase the message sending intervals until the value of intervalselector reaches 4. At this point the retransmission interval is 20 seconds, and the algorithm continues indefinitely at this rate. A detailed, step-by-step walkthrough of the algorithm can be found in Appendix 4.

[21] The keep-alive message is a retransmission of the latest GOOSE message, continuously transmitted between long time intervals, typically 20 seconds. It essentially informs other IEDs that "the sender is still alive". See Section 4.8 or IEC 61850-8-1 for more details.

```
int main() {

    initialise_iec61850();
    s = createSocket(&socket_address);
    int i = 0;

    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.datasetDecodeDone =
    &GSEcallbackFunction; // prototype

    int milliintervals[] = {1024,   1024,   4098,   16384,  32768};
    int ticksintervals[] = {0,      0,      4,      20,     40   };

    for(;; ticks++)
    {
        // Receive
        do {
            len = recvfrom(s, buf, BUFFER_LENGTH, 0, NULL, NULL);
            if(len){
                gse_sv_packet_filter(buf, len);
            }
        } while(len >= 0);

        if( ticks >= ticksintervals[intervalselector] )
        {
            //send old
            if( intervalselector < 4 )
            {
                intervalselector++;
            }
            len = Laptop.S1.C1.LN0.Status.send(buf, 0,
            milliintervals[intervalselector]);
            strncpy(socket_address.sll_addr,
            Laptop.S1.C1.LN0.Status.ethHeaderData.destMACAddress, ETH_ALEN);
            send_result = sendto(s, buf, len, 0,
            (struct sockaddr*)&socket_address, sizeof(socket_address));
            if (send_result == -1) {
                fprintf(stderr, "Send failed");
            }
            printf("Package sent, timeAllowedToLive: %i\n",
            milliintervals[intervalselector] );
            fflush(stdout);
            ticks = 0;
        }
        usleep(500000);
    }
    return 0;
}
```

Figure 22. The GOOSE retransmission scheme implemented in main(). The algorithm is based on a platform-specific sleep function which creates the delays between retransmissions of the same GOOSE message. One loop of the infinite for loop corresponds to a delay of approximately 0.5 seconds. The algorithm uses an array the values of which give the number of loops that should be performed between retransmissions to create the required delays.

3.4.3. The message exchange

The application, running on the LDU, copies the binary value received in a GOOSE message from Vamp 50 over to data attribute stVal in logical node DCIPa. That data attribute is referenced by DataSet Status, and is then sent back as a GOOSE message to Vamp 50. In other words, the application simply returns the received value to the sender.

The application begins by retransmitting a GOOSE message, even before receiving a GOOSE message first, at the lowest retransmission time interval. When a GOOSE message is received and decoded, the GSEcallbackFunction is executed. The callback function is depicted in Figure 23. A message indicating that the F1 button on the front panel of Vamp 50 was pressed, along with the timeAllowedToLive value of the received message, is printed to console. If the received value was 1, the string indicates that the button is on and otherwise off.

The following if statement checks whether the received value is the same value that is stored in data attribute stVal in logical node DCIPa of IED Laptop. If the values are different, it means that an internal event has occurred in Vamp 50 (button F1 or F2 has been pressed), and the if statement evaluates as true. The old value in stVal is then overwritten with the received value. Intervalselector and ticks are reset to 0 indicating that a new retransmission starting from the shortest retransmission interval should begin after that the application exits the callback function. A new GOOSE message with the shortest retransmission interval is then generated and sent over the network, and a message indicating that the message was sent is printed to console.

If the received value equals the value stored in stVal in logical node DCIPa, the received message is assumed to be a retransmission of an earlier internal event and will be ignored. A more sophisticated method would be to check the stNum parameter of a GOOSE message, which increments by one for each internal event. The parameter sqNum is incremented by each retransmission of the same event.

```
void GSEcallbackFunction(CTYPE_INT32U timeAllowedToLive, CTYPE_TIMESTAMP T,
        CTYPE_INT32U stNum, CTYPE_INT32U sqNum)
{
    printf("Button F1: %s, timeAllowedToLive: %i\n",
    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.Relay_VI1GGIO_137
    _SPCSO_stVal == 1 ? "On" : "Off",timeAllowedToLive);

    if( Laptop.S1.C1.DCIPa_1.EngOnOff.stVal !=
    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.Relay_VI1GGIO_137_
    SPCSO_stVal )
    {
        Laptop.S1.C1.DCIPa_1.EngOnOff.stVal =
        Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.
        Relay_VI1GGIO_137_SPCSO_stVal;

        intervalselector = 0;
        ticks = 0;

        len = Laptop.S1.C1.LN0.Status.send(buf, 1, 1024);

        strncpy(socket_address.sll_addr,
        Laptop.S1.C1.LN0.Status.ethHeaderData.destMACAddress, ETH_ALEN);

        send_result = sendto(s, buf, len, 0,
        (struct sockaddr*)&socket_address, sizeof(socket_address));

        if (send_result == -1) {
            fprintf(stderr, "Send failed");
        }

        printf("Package sent, timeAllowedToLive: 1024\n");
    }
}
```

Figure 23. The GSE callback function. The function simply returns the value of a data attribute received in a new GOOSE message to the sender.

3.4.4. Testing and results

Results obtained from the PuTTY terminal are depicted in Figure 24. The figure shows how the LDU, at start-up, starts the retransmission with the shortest interval. The value of the timeAllowedToLive parameter for each message is printed out. The interval grows larger until the keep-alive interval is reached (20 seconds) with the value of the timeAllowedToLive parameter being 32768 milliseconds. Vamp 50 transmits its own messages at a keep-alive interval of 20 seconds. The value of the timeAllowedToLive parameter for keep-alive messages being retransmitted at this rate is 40000

milliseconds[22]. The application does not measure the time elapsed between the received messages or check whether this time window is smaller than the timeAllowedToLive value. As Figure 24 shows, the value of the data attribute stVal, which relates to button F1, is initially 0 (Off).

Message	timeAllowedToLive:
Package sent,	1024
Package sent,	4098
Button F1: Off,	40000
Package sent,	16384
Package sent,	32768
Button F1: Off,	40000
Package sent,	32768
Button F1: Off,	40000
Package sent,	32768
Button F1: Off,	40000
Button F1: On,	10
Package sent,	1024
Button F1: On,	10
Button F1: On,	10
Button F1: On,	16
Button F1: On,	32
Button F1: On,	64
Button F1: On,	128
Package sent,	1024
Button F1: On,	256
Button F1: On,	512
Button F1: On,	1024
Package sent,	4098
Button F1: On,	2048
Button F1: On,	4096
Package sent,	16384
Button F1: On,	8192
Button F1: On,	16384
Package sent,	32768
Button F1: On,	32768
Button F1: On,	40000
Package sent,	32768

Figure 24. The message exchange between the LDU and Vamp 50, monitored by PuTTY. An internal event in Vamp 50 changes the value of button F1 to On and the message is retransmitted multiple times. LDU receives the changed value and returns it in its own retransmissions.

[22] This means in practice that every second packet sent could be lost without this being noticed by the subscriber. The values are however adjustable.

After a few initial keep-alive retransmissions, the button F1 is pressed and Vamp 50 lets out a burst of rapid retransmissions. The retransmission intervals grow until the keep-alive rate is achieved. The LDU replies to the first message received after that the button was pressed by sending out its own burst of rapid retransmissions.

Figure 25 depicts the contents of the GOOSE messages of Vamp 50 and LDU, respectively, sent before (A) button F1 was pressed and after (B) button F1 was pressed. Between A and B, the Boolean data attribute[23] value has changed from False to True and the value of the parameter stNum has increased by one. The results show that the communication works, and the requirements have therefore been met.

[23] The DataSet of IED VAMP actually contained two data attributes, but only one was used.

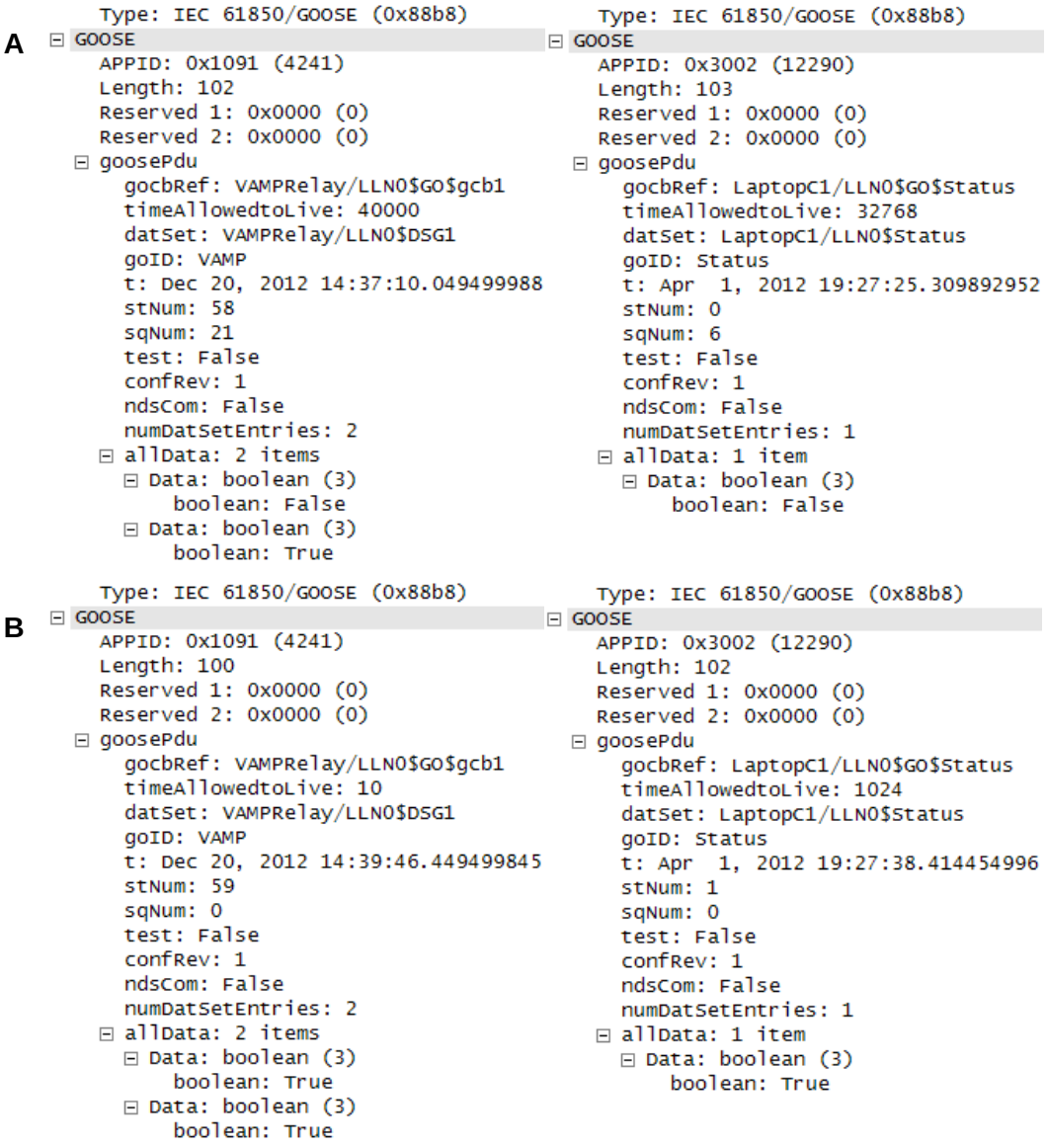


Figure 25. Contents of the GOOSE messages transmitted by Vamp 50 and the LDU, respectively. A = messages sent at the stable keep-alive retransmission rate. B = the first messages transmitted after button F1 was pressed. (Screenshots from Wireshark.)

4. THE ABSTRACT COMMUNICATION SERVICE INTERFACE

The Abstract Communication Service Interface (ACSI) is a layered communication architecture that provides for systems in power utility automation a virtual model of the real devices and their data, i.e., a virtual interface to the analogue world. The virtual model is made visible and accessible through ACSI services, which are used to exchange the information represented by the data objects and their data attributes between different devices according to well-defined rules described in IEC 61850-5. (IEC 61850-7-2 2010: 11; IEC 61850-7-1: 16-18, 72-73.)

The ACSI provides abstract definitions of

- hierarchical information modelling classes,
- services operating on instances of these classes, and
- parameters associated with each service.

The ACSI enables interoperability between different devices by abstracting away the underlying implementation details. The class models and services are thus independent of any specific protocol stacks, implementations, and operating systems. Realization of any concrete information exchange requires mapping of the abstract class models and services to a real protocol stack through Specific Communication Service Mappings (SCSMs). (IEC 61850-7-2 2010: 11.)

Due to the abstraction, only those aspects of a real substation device or a real function that are visible and accessible over a communication network are modelled, which leads to the hierarchical class models. Conversely, anything that is not modelled as a class of a service, logical device, logical node, data object, data attribute, control block, etc. is not visible to the network. The conceptual class model of the ACSI is a comprehensive model for how physical devices shall organize their data. The ACSI class model describing a real substation device is repeated for convenience in Figure 26. (IEC 61850-7-2 2010: 11; IEC 61850-7-1 2011: 80; Mackiewicz 2006.)

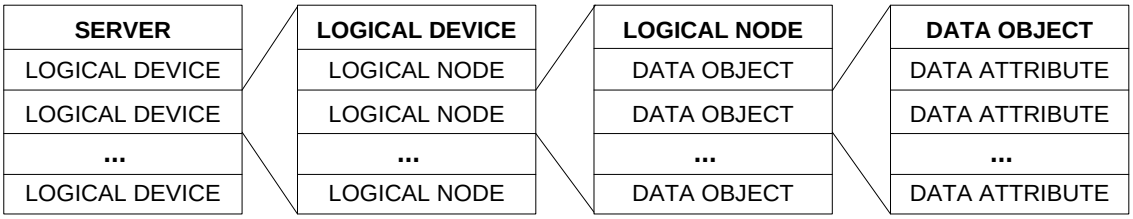


Figure 26. The conceptual class model of the ACSI. (Adapted from Liang & Campbell 2008.)

Each individual class model of the ACSI class model comprises a number of *attributes* and services. When a class is instantiated, it becomes an object. The services of the object can then be used by a client—mainly to write information to, or read information from the object using a client/server communication model. The attributes describe or characterize the externally visible features of an instance of a class. Typical class attributes are the object name and the object reference, which represent the name and path-name of an instance of the class, respectively. Object names and references are described in more detail in Section 4.3. Other attributes are the object classes that the class contains, for example logical nodes, data objects, data attributes, and control blocks. Most interactions with a physical device are through data objects and data attributes in logical nodes and services. (IEC 61850-7-2 2010: 17, 40; IEC 61850-7-1 2011: 54, 87; Liang & Campbell 2008.)

The object classes defined by the class model are (IEC 61850-7-2 2010: 14, 17, 45; IEC 61850-7-1 2011: 14, 64.):

- **Server** - represents the externally visible behaviour of a physical device. All other ACSI models are part of the server.
- **Logical device** - mainly a composition of logical nodes and additional services. Represents the information produced and consumed by a group of typical domain-specific automation, protection or other application functions.
- **Logical node** – represents the information produced and consumed by a single domain-specific automation, protection, or other application function.

- **Data objects** - represents domain specific information of applications located in a physical device, for example status or measurement.

The service modelling classes of the ACSI provide services operating on instances of data objects, data attributes, DataSets, and other attributes that are contained in logical nodes. All services are requested by applications (clients) and responded by servers. The services and the response to those services are well defined, which enables all physical devices to behave in an identical manner seen from the network behaviour perspective. Due to the abstraction, only those aspects that are needed in describing the required actions on the receiving and sending side of a service request are defined. (IEC 61850-7-2 2010: 18; IEC 61850-7-1 2011: 24, 31-32; Liang & Campbell 2008; Mackiewicz 2006.)

Many services may be used to remotely manage IEDs, e.g. to define DataSets, to set a reference to a specific value, or to enable sending specific reports by a report control block. (IEC 61850-7-1 2011: 32.)

The service modelling classes are listed below (IEC 61850-7-2 2010: 18.):

- **Application association** – provides mechanisms for establishing and maintaining connections between devices.
- **DataSet** – permits the grouping of data objects and data attributes.
- **Substitution** – forces a specific data attribute to be set to a value independent of the process.
- **Setting group control** – defines how to switch from one set of setting values to another one and how to edit setting groupings.
- **Control blocks for reports and logs** – describes the conditions for generating reports and logs based on parameters set by configuration or by a client.
- **Control blocks for generic substation events** – supports a fast and reliable system-wide distribution of GOOSE messages controlled by the GOOSE control block.
- **Control block for transmission of Sampled Values** – fast and cyclic transfer of sampled data values.

- **Control** – describes the services to control.
- **Time and time synchronization** – provides the time base for the device and system.
- **File system** – defines the exchange of large data blocks.
- **Tracking** – provides a diagnosis interface to track services.

Control blocks are modelled similar to data classes, but there is a fundamental difference between instances of data classes and control blocks: objects are used to interface to application level functions, while control blocks configure the communication services. Control blocks have attributes that can be set to specific values through services or SCL files. The values of the attribute-set define the dynamic behaviour of the control block. All control block attributes can be read by other IEDs. (IEC 61850-7-1 2011: 35; IEC 61850-7-2 2010: 70.)

The service parameters are not a part of a data model—they represent information conveyed by services between clients and servers. For example, a client may use the GetServerDirectory service to retrieve a list of the names of all the logical devices contained in a Server object. The service parameters of the GetServerDirectory service are given in Table 4. The parameter ObjectClass of the request contains an identification of the selected class. The selected class in this example is "logical-device". The parameter Responce+ indicates that the request was successful, and the server returns the parameter Reference [0...n] containing the object references of the logical devices to the client. The parameter Response- indicates that the service request failed, and the server returns the parameter ServiceError containing an error message like "parameter-value-inappropriate" to the client. (IEC 61850-7-2 2010: 31; IEC 61850-7-1 2011: 22.)

Table 4. Service parameters of the GetServerDirectory service of the Server class. (IEC 61850-7-2 2010: 31.)

Parameter name
Request
ObjectClass
Response+
Reference [0..n]
Response–
ServiceError

IEC 61850 outlines two main groups of communication models: the client/server model and the publisher/subscriber (peer-to-peer) model. The client/server model is used for ACSI services, and the publisher/subscriber model is used for communication services such as GOOSE and Sampled Values. The two communication methods are depicted in Figure 27. (Ozansoy 2006: 120-121.)

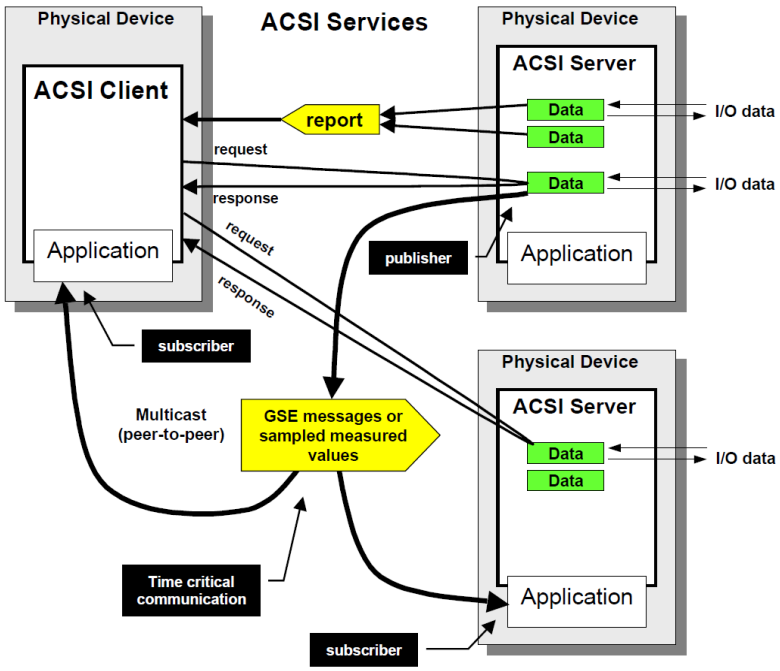


Figure 27. Two groups of ACSI communication methods: one group uses a client/server model with services like control or get data values. The second group uses a peer-to-peer model with GSE (GOOSE) services and Sampled Value services. (IEC 61850-7-1 2011: 70-71.)

Some of the classes used for information modelling are presented in the following sections. The more advanced aspects of data modelling, for example the *SubDataObject* classes and the *SubDataAttribute* classes with their recursive properties are beyond the scope of this thesis and therefore omitted.

4.1. The application association model

The application association model is used to establish communication between devices, i.e., a conveyance path between a client and a server for the exchange of messages. The conceptual model comprises both class definitions of application associations, two-party and multicast, and access control concepts—a mechanism for controlling the access to the instances of a device. (IEC 61850-7-2 2010: 32; IEC 61850-2 2003: 5.)

An application (client) must first establish a valid two-party application association (TPAA) with a server before it can request a service. The TPAA class provides a bi-directional connection-oriented information exchange by conveying service requests and responses, both confirmed and unconfirmed services, between client and server, as illustrated in Figure 28. The TPAA thus provides a virtual view of the server to the application, and this view defines the visibility of objects in the server from the application point of view. It also defines what kinds of services of those objects are accessible from the application. (Liang & Campbell 2008; 61850-7-2 2010: 32.)

The multicast application association (MCAA) class conveys unidirectional, unconfirmed messages (in one direction only) between one source (publisher) and one or many destinations (subscribers). Examples of MCAAs are GOOSE and SV messages. The MCAA ceases to exist as soon as the service has been processed. (IEC 61850-7-2 2010: 32, 37.)

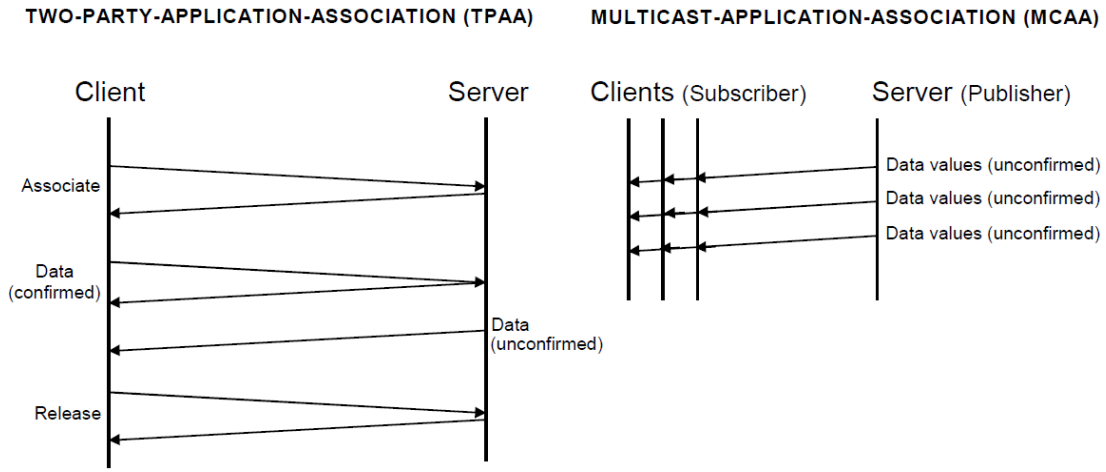


Figure 28. Operational principles of the two-party (TPAA) and multicast application associations (MCAA). The TPAA is used to establish a connection oriented association between client and server. The MCAA is used to establish a connectionless association between publisher and subscriber(s). (IEC 61850-7-2 2010: 33, 37.)

A typical interaction between client (application) and server in TPAA (depicted on the left side of Figure 28) as presented in (Liang & Campbell 2008) proceeds as follows:

1. The client establishes a TCP connection with the server.
2. The client "logs in" to the server by requesting an appropriate service and by providing authentication related information as parameters.
3. The server validates the information provided by the client and creates a TPAA object, which provides a virtual view of the server to the client.
4. The client requests subsequent services while the server processes the requests and responses with appropriate responses defined in the IEC 61850 standard.
5. The client issues a release request to the server.
6. The server reclaims the TPAA of the application and ends the session.

4.2. Mapping of an abstract interface to a concrete interface

The abstract class models and service models of the ACSI become usable only when mapped to a concrete “on-the-wire” transport protocol. This is done through a Specific Communication Service Mapping (SCSM). (Ozansoy 2006: 37.)

The development in communication technology progresses faster than the development of the functionalities in substation automation. This means that the currently used protocol stack might get replaced by a new state-of-the-art protocol stack in the near future. To accommodate this change, the abstract class models and service models of the ACSI were designed to be independent from any particular protocol stack. The capability of the ACSI to adapt to different protocol stacks, without having to make any changes to the information models or databases, is usually seen as a significant advantage of IEC 61850. (De Mesmaeker, Rietmann, Brand & Reinhardt 2005; Mohagheghi, Tournier, Stoupis, Guise, Coste, Andersen, & Dall 2011; Ozansoy 2006: 3.)

The mapping process involves the implementation of the standard’s object models by using the existing models of an underlying communication service. The concrete information exchange, i.e. the specific format, encoding and transmission of the messages that carry the service parameters of a service is defined in the SCSM. IEC 61850-8-1 specifies two different SCSMs for the object models and services of the ACSI (IEC 61850-7-1 2011: 24; Ozansoy 2006: 37; IEC 61850-8-1 2011: 14):

- mapping of the object models and service models to the Manufacturing Message Specification (MMS) protocol that operates over the TCP/IP and Ethernet stacks, and
- direct mapping of GOOSE messages onto Ethernet frames at the DataLink layer.

An implementation that conforms to a specific communication profile implements the application association model required for that profile. The relations between the application associations specified in IEC 61850-7-2 and the communication profiles specified in IEC 61850-8-1 are listed in Table 5.

Table 5. Communication profiles and the application association models that are supported. (IEC 61850-8-1 2011: 58.)

Communication profile	ACSI association model(s) supported
Client/server	Two party
GSE Management	Two party
GOOSE	Multicast
GSSE	Multicast
Time Sync	Two party or multicast

The client/server communication profile maps the ACSI TPAA model to the MMS protocol, as shown in Figure 29. This mapping defines how the concepts, objects and services of the ACSI are implemented using MMS concepts, objects and services. The TCP protocol is used for connection-oriented communication, which guarantees the message delivery. An introduction to the MMS protocol and the SCSM to MMS can be found in Appendix 5.

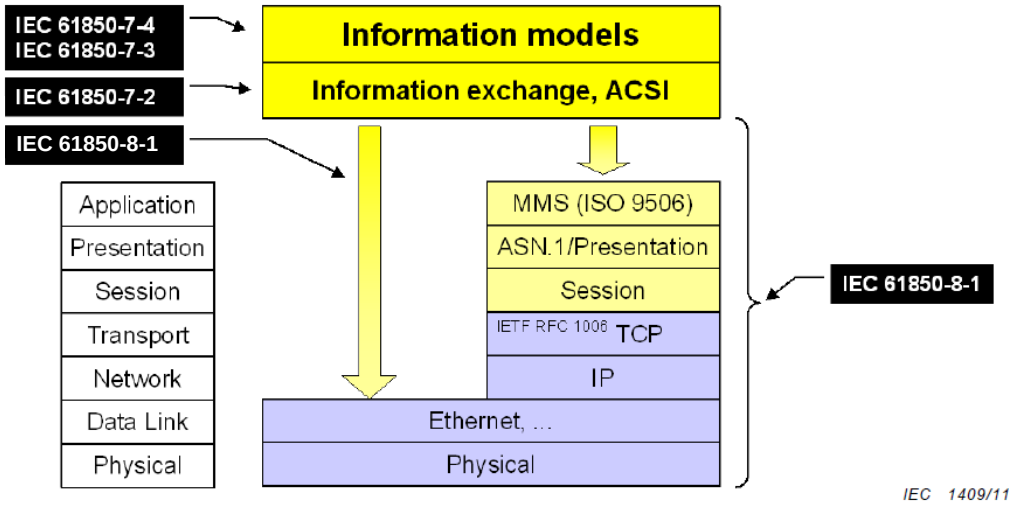


Figure 29. The abstract models of the ACSI need to be mapped to a concrete protocol stack through an SCSM. The data models and service models are mapped to the Application layer Manufacturing Message Specification (MMS) protocol which operates over TCP/IP and Ethernet. GOOSE messages are mapped directly onto Ethernet frames on the DataLink layer. (Adapted from IEC 61850-7-1 2011: 25.)

The GOOSE communication profile maps the ACSI MCAA model directly onto Ethernet frames on the DataLink layer. The direct mapping avoids any delays caused by the execution of middle layers in the protocol stack, making the GOOSE protocol suitable for time-critical data transfer. The drawback of connectionless communication is that the message delivery is not guaranteed—the publisher does not know which IEDs will receive the message, and the receiving IEDs will not acknowledge the successful reception of a message. The subscribing IEDs must cope with message loss, message duplication, messages delivered out-of-order and the loss of connectivity. In other words, connectionless multicast communication is unreliable. The SCSM for GOOSE is described in Section 4.8. (Xyngi & Popov 2010; Hou & Dolezilek 2008; IEC 61850-8-1 2011: 14, 58.)

The application and transport profile layers used in the SCSMs for MMS and GOOSE can be found in Appendix 1.

4.3. Object names and references

IEC 61850 differentiates between object names and object references. An object name (class instance name) identifies an instance of a class at one particular hierarchy level. An object reference (semantic) is constructed from the full path-name of the object. For example, the names of the logical device, logical node, data object, and data attribute are concatenated into a full path-name that identifies the instance of a data attribute uniquely. The example is illustrated in Figure 30. A logical node name always contains an instance number as a suffix (e.g. 1), and may also contain a prefix (e.g. Q0). (IEC 61850-7-1 2011: 52, 93; IEC 61850-7-2 2010: 24.)

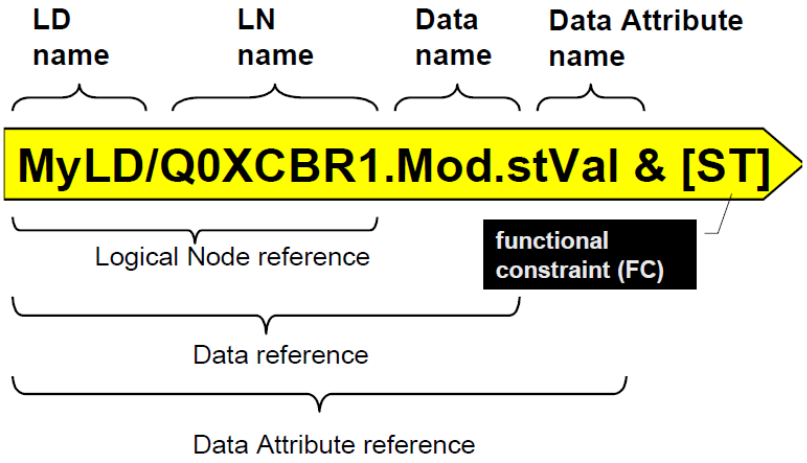


Figure 30. An example of object names and references. The figure shows the complete object reference (data attribute reference) to data attribute stVal. (IEC 61850-7-1 2011: 93.)

The logical device names are not specified by the standard, but they have to be unique in a particular subnetwork. Data object names and data attribute names, from IEC 61850-7-4 and IEC 61850-7-3 respectively, shall be used unchanged: a collection of standardized, abbreviated data object names for substation functions and equipment can be found in IEC 61850-7-4, and the semantics of all data attribute names can be found at the end of IEC 61850-7-3. The semantics of the service models with their attributes and the semantics of the services that operate on these attributes, including the parameters that are carried with the service requests and responses, are defined in IEC 61850-7-2. (IEC 61850-7-1 2011: 22, 24, 52, 93-94; Mohagheghi, Tournier, Stoupis, Guise, Coste, Andersen & Dall 2011; Mackiewicz 2006.)

4.4. The server class model

A physical device may contain zero or more servers, usually one. The server has two roles: to communicate with a client and to send information to peer devices. Clients issue service requests and receive confirmations of the service that has been processed in the server. All service requests and responses are communicated by the protocol stack that is being used by a specific SCSM. A server is instantiated from the

GenServerClass and has a service access point[24] (SAP) attribute, through which the logical device(s) of the physical device connects to a subnetwork. An SAP can be described as a logical representation of a network interface card, and it can be used without a server. The *client logical nodes* of any logical device can access a subnetwork through a SAP. Each server contains one or more logical devices with logical nodes. These logical nodes can, as *server logical nodes*, be accessed by a client logical node through a SAP. The client/server roles and the SAP are illustrated in Figure 31.

Logical nodes can play the role of clients or servers, or both at the same time. As an example, a switch controller logical node (CSWI) may receive data as a client from a process bus, and provide data as a server to the station bus. (Ozansoy 2006: 88, 94; IEC 61850-7-2 2010: 17, 29; IEC 61850-2 2003: 5; Liang & Campbell 2008; IEC 61850-7-1 2011: 76-77; IEC 61850-6 2009: 24, 57.)

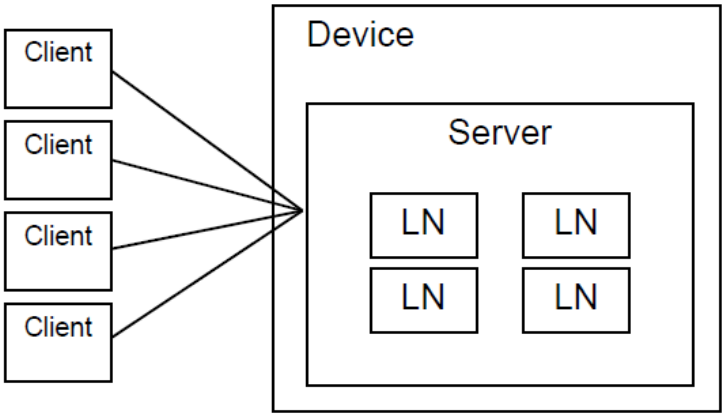


Figure 31. Client/server and logical nodes. Client logical nodes in a logical device in one IED may access the server logical nodes in a logical device belonging to a server in another IED through the server's SAP. The logical nodes residing in any logical device in the IED may, as clients, access the subnetwork through a SAP. (IEC 61850-7-1 2011: 77.)

[24] Each server has only one, logical, access point. Each access point of an IED to a communication bus is uniquely defined. (IEC 61850-2 2003: 5.)

The SAP is generally an abstraction of an IP address and identifies the server within the scope of a subnetwork. It is a logical construct through which a peer selects a communication protocol or access to an application. The SAP is required by most services to address the server. As an example, at least one SAP needs to be supported for GOOSE services, where the SAP is defined as the combination of the physical MAC address, Ethertype, and Application ID specified by the profile for GOOSE. Implementations that use the client/server profile shall support at least one PresentationAddress (Presentation layer address) that makes use of the TCP/IP T-Profile. The *type* of the SAP thus depends on the SCSM used for the service—the SAP is a profile[25] of the underlying SCSM. (IEC 61850-2 2003: 22; IEC 61850-7-2 2010: 29-30; Ozansoy 2006: 92; IEC 61850-8-1 2011: 56.)

The SAPs (communication addresses/interfaces) of a subnetwork are used for building application associations between logical nodes. Client logical nodes need to connect to a SAP prior to establishing a TPAA with the server's logical nodes, and subscriber logical nodes need to establish an MCAA with publisher's logical nodes. Logical nodes are the only objects in IEC 61850 that communicate with each other. (IEC 61850-7-1 2011: 77; Liang & Campbell 2008; IEC 61850-6 2009: 24-25.)

Logical nodes may as clients use all SAPs of a physical device to access logical nodes on other physical devices. An SAP needs to have a server if the physical device needs to be supervised remotely, as the LLN0 and LPHD of the server's logical device are used to supervise and control the physical device. A physical device can manage without a server if all its logical nodes use an SAP as clients only and no remote supervision is required. (IEC 61850-6 2009: 24, 57.)

Although only one service, the GetServerDirectory, is specified in the GenServerClass definition, several GetXXDirectory and GetXXDefinition services

[25] The selection of the entire seven layers of a service access point represents a communication profile. (IEC 61850-2 2003: 22.)

are specified in IEC 61850-7-2 to support the self-description of a device. A client may use these services to make inquiries regarding the names and definitions of the objects that the server contains, i.e., the client can retrieve the definition of the complete hierarchy and the definition of all accessible information of all instances of all underlying classes in a given server. The operation of these services is illustrated in Figure 32. (IEC 61850-7-2 2010: 29-31.)

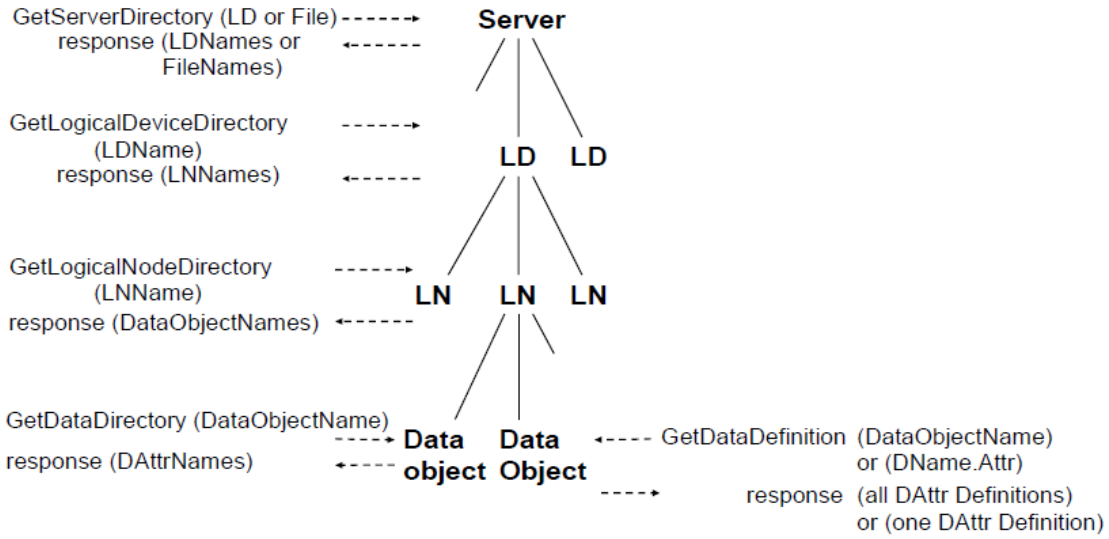


Figure 32. The requests and responses of the GetXXDirectory and GetXXDefinition services of the server. These services are used by clients to retrieve the object names, references, and definitions, or the complete self-description of a given device. (IEC 61850-7-2 2010: 30.)

4.5. The logical node class model

Logical nodes for common applications are instantiated from compatible logical node classes, defined in IEC 61850-7-4. For example, the compatible logical node class XCBR represents common information of a real circuit breaker, and this class can be reused for circuit breakers of various brands and types. The compatible logical node classes are application-specific specializations of the basic GenLogicalNodeClass, defined in IEC 61850-7-2, from which they inherit their structure and all definitions.

The relationships between different logical nodes are shown in Figure 33. The *common logical node class* provides data objects which are mandatory or conditional to all compatible logical node classes. (IEC 61850-7-1 2011: 28, 91; IEC 61850-7-4 2010 21-22.)

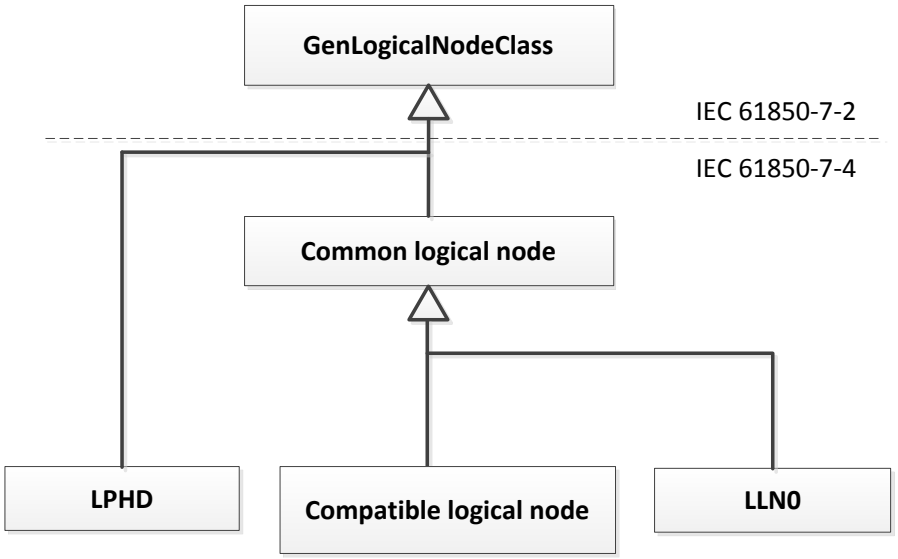


Figure 33. Logical node relationships. The LPHD and the common logical node are independent of the application domain. All other logical nodes are domain-specific and inherit at least the mandatory data objects of the common logical node class. The common logical node itself is never instantiated. (Adapted from IEC 61850-7-4 2010: 21.)

A logical node groups several *data object classes* to build up a specific functionality. For example, XCBR contains over 15 data object classes that represent meaningful information of the application. Most of the information defined in a logical node is therefore described by the data object classes it consists of. (IEC 61850-7-1 2011: 28, 80.)

All logical nodes have a specific name and reference attribute, and one or more data object classes. The data object classes in a compatible logical node are given by the common logical node. All logical nodes may additionally contain zero or more of the following classes (IEC 61850-7-2 2010: 40):

- Buffered Report Control Block,
- Unbuffered Report Control Block, and
- DataSet.

Compatible logical nodes may also support zero or more of the following classes:

- GOOSE Control Block,
- Setting Group Control Block,
- Multicast Sampled Value Control Block,
- Unicast Sampled Value Control Block,
- LOG, and
- Log Control Block.

The LLN0 and LPHD are two special infrastructure logical nodes. The LPHD is used for accessing hardware related data of an IED. It represents the common data of the physical device hosting the logical device, and contains three mandatory data objects:

- PhyNam represents the physical device name plate,
- PhyHealth represents the physical device health, and
- Proxy is used as an indication of whether the logical node is used as a proxy.

The LLN0 is used for accessing logical device related data of an IED. A lot of information required for device management is modelled in IEC 61850-7-4 as data object classes in LLN0. (Ozansoy 2006: 43; IEC 61850-7-1 2011: 64, 80; IEC 61850-7-4 2010: 22.)

4.6. Data object classes

Data objects are instantiated from the basic GenDataObjectClass, which was earlier referred to as DATA class in the first edition of IEC 61850-7-2. However, which data attributes that GenDataObjectClass should contain is not specified. (IEC 61850-7-1 2011: 80; Liang & Campbell 2008.)

A data object is basically a container consisting of a set of data attributes which relate to the same application. Data objects thus facilitate the managing and exchange of the values of the data attributes of a particular application. The data attribute classes that are shared by many data object classes for the most common applications have been grouped together for convenience. These data attribute classes, common to many data object classes, are called common data classes (CDCs). The CDC is thus a refinement of the GenDataObjectClass, to which the previously missing data attributes have been added. The core CDCs are classified into the following groups**:**

- status information,
- measuring information,
- controllable status information,
- controllable analogue information,
- status settings,
- analogue settings, and
- description information.

A data object is instantiated from the GenDataObjectClass, to which the data attributes of a particular CDC has been assigned. The data attributes that the data object instance contains are thus instances of the data attribute classes of the CDC. The chosen CDC describes the type and structure of the instantiated data object, i.e., the data object is *typed* by the CDC, the type being defined by the functionality of the data attributes. Data objects that are typed by a specific CDC (i.e., have been assigned application-specific data attributes) are referred to as *compatible data objects*. Many compatible data objects for a wide range of well-known applications are defined in IEC 61850-7-4 and IEC 61850-7-3, where they are assigned suitable application specific names. The compatible data objects do not usually implement all the data attribute classes of their CDC—they are thus specialized (further refined) CDCs that often inherit only the data attributes which are required (mandatory) for instantiation. The refinement of the GenDataObjectClass to a CDC, and further refinement of the CDC to a compatible data object class is illustrated in Figure 34. (Liang & Campbell 2008; Mackiewicz 2006;

IEC 61850-7-1 2011: 23, 28, 30, 80-82, 87; IEC 61850-7-4 2010: 10; IEC 61850 7-2 2010: 45, 51.)

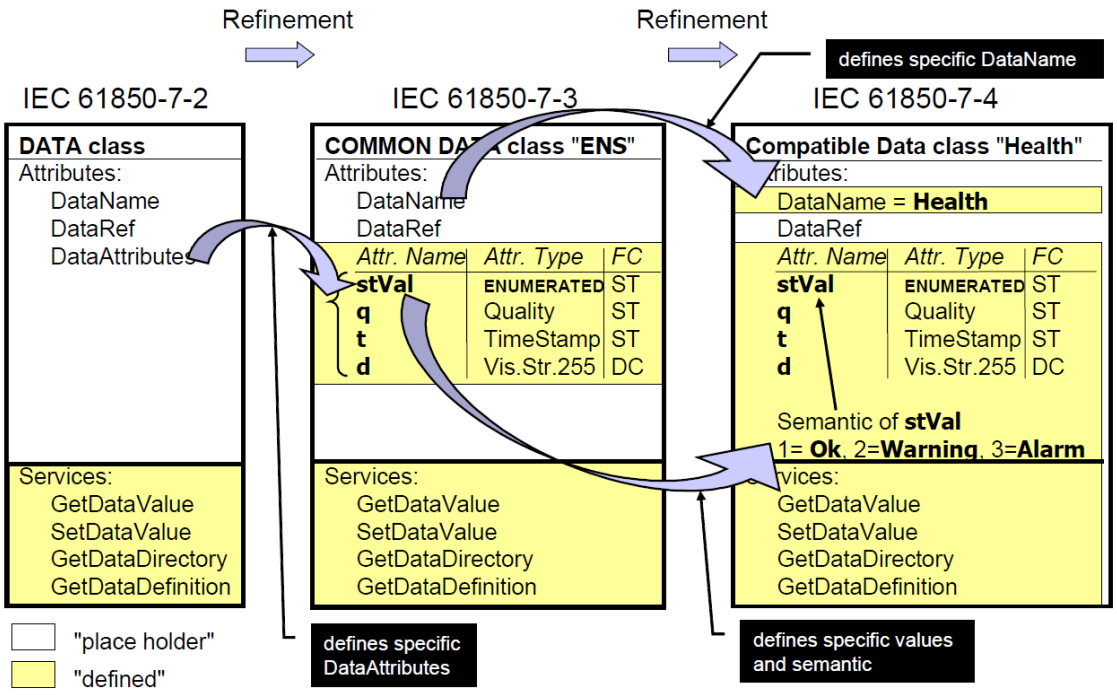


Figure 34. Refinement of the GenDataObjectClass (referred to as DATA class in the figure) to a CDC, and further refinement of the CDC to a compatible data object class with defined data attributes and attribute values. (IEC 61850-7-1 2011: 80.)

As an example, illustrated in Figure 35, the logical node myXCBR1 is instantiated from the compatible logical node class XCBR which comprises multiple compatible data object classes (e.g. Pos) each refined from a CDC (e.g. DPC), where the whole set of all data attributes (e.g. stVal, q, and t) inherited by a compatible data object class is (a subset of) a CDC. As a compatible data object class usually only implements a subset of the data attributes (i.e., the mandatory data attributes) of a CDC, the Type attribute of the compatible data object class is assigned a name (value) with an arbitrary prefix in front of the CDC name, for example simpleDPC. (IEC 61850-7-4 2010: 105-106.)

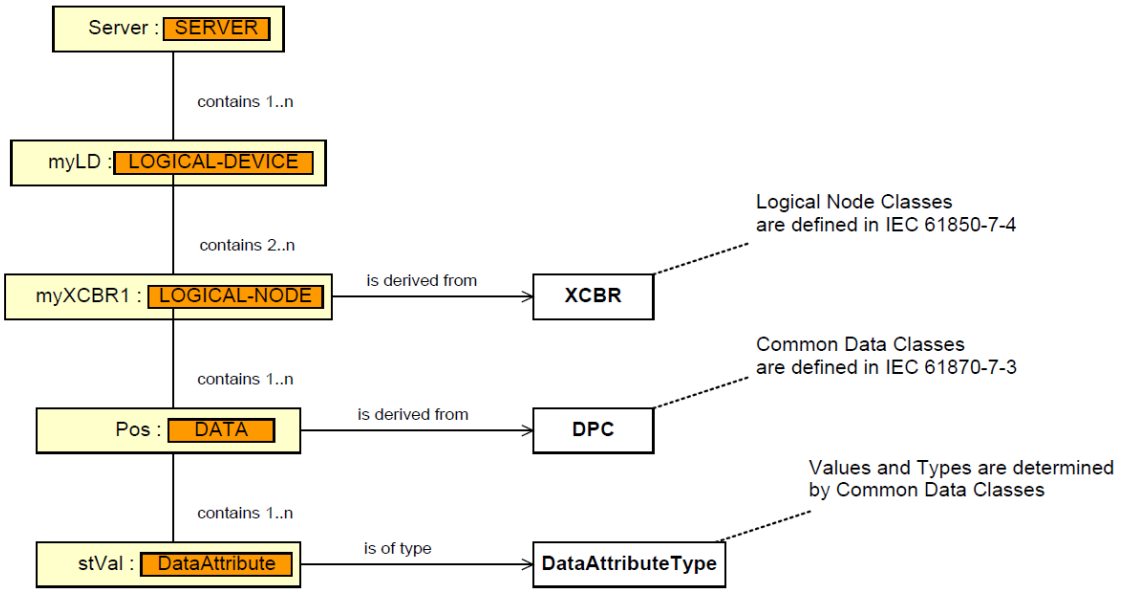


Figure 35. Abstract data model example. The logical node myXCBR1 is instantiated from compatible logical node class XCBR. The compatible data object Pos is refined from common data class DPC. The data attribute stVal has a type and value determined by the CDC it is inherited from. (IEC 61850-7-1 2011: 87.)

Each CDC has services assigned to it that define the possible services that are allowed to be operated on the data attributes. The functional constraint (FC) is a property of a data attribute that indicates the services for example read value, write value, substitute value, etc. that may be applied to that data attribute. The data attributes of a CDC are also grouped into categories by their FCs. (Mackiewicz 2006; IEC 61850-7-1 2011: 54; IEC 61850-2 2003: 11.)

4.6.1. The functional constraint

The functional constraint (IEC 61850-7-2 2010: 53; IEC 61850-7-1 2011: 89; Liang & Campbell 2008):

- characterizes the specific use of a data attribute,
- functionally restricts data attributes; groups all the data attributes by functions instead of grouping them by data objects,

- can be seen as a filter of the data attributes,
- indicates which services can be used to access the values of the data attributes,
- is used in the definition of data objects, and
- is a property of the GenDataAttributeClass model.

The functional constraint values are defined in IEC 61850-7-2 along with the allowed services. The FCs are as follows (IEC 61850-7-2 2010: 54):

- **ST –** Status information
- **MX –** Measurands (analogue values)
- **SP –** Setting (outside setting group)
- **SV –** Substitution
- **CF –** Configuration
- **DC –** Description
- **SG –** Setting group
- **SE –** Setting group editable
- **SR –** Service response
- **OR –** Operate received
- **BL –** Blocking
- **EX –** Extended definition (application name space)
- **XX** – Representing all data attributes as a service parameter

Functional Constrained Data (FCD) is a reference to an ordered collection of data attributes having the same FC value. These data attributes all reside in a same data object. The data attributes of the FCD are arranged in the same order as they appear in the data object. A Functional Constrained Data Attribute (FCDA) is a reference to a single SubDataObject, data attribute, or SubDataAttribute having a specific FC value. FCDs and FCDAs are used, for example, in remotely created DataSets. (IEC 61850-7-2 2010: 53, 55.)

4.7. The DataSet class

The reporting and logging functions as well as the SV and GOOSE messaging functions utilize DataSets for their spontaneous data transmission. The object references (FCDs and FCDAs) of functionally constrained data objects, SubDataObjects, data attributes, and SubDataAttributes can be grouped into a DataSet. The DataSet is thus an ordered collection of FCDs and FCDAs. The object references that the DataSet contains are called the members of the DataSet, and can reference any objects on the same server where the DataSet itself is defined. The functionally constrained objects in a server may be referenced by one or more DataSets.

DataSets that are referenced by a control block must reside in the same logical node as the control block itself. This means that all GOOSE and SV data flow definitions must reside in the LLN0. When a GOOSE control block needs to transmit a message, the name of the referenced DataSet and the values of the objects referenced by that DataSet are encoded into a GOOSE message. The references themselves are normally not included in the message—from this follows that the members of the DataSet as well as the order these members appear in must be known by both the publisher and the subscriber(s). The client or subscriber(s) may later retrieve the object references out of the DataSet definition. (IEC 61850-7-2 2010: 61, 63; IEC 61850-7-1 2011: 43; IEC 61850-6 2009: 26.)

DataSets may be created or configured through the CreateDataSet service and the members of a DataSet and their order can be retrieved by the GetDataSetDirectory service. A DataSet instance may be persistent or non-persistent, which relates to the visibility and sustainability of the DataSets. A persistent DataSet, or preconfigured DataSet, is visible to clients of any TPAA, and is not deleted when the TPAA over which it is created is released. A persistent DataSet can be referenced by any control block class, for example the GOOSE control block. A non-persistent DataSet is visible only to the client that created the instance, and is automatically deleted when the TPAA which it is created over is released. A non-persistent DataSet can be referenced only by an unbuffered report control block or a unicast sampled value control block. (IEC 61850-7-2 2010: 61-62.)

A control block that references a specific DataSet monitors the objects that the DataSet references. An *internal event* is created when a value change, according to some trigger condition specified by the attribute TrgOp, occurs in any of the monitored objects. The internal events are used to trigger reporting, logging, transmission of SV messages, and transmission of GOOSE messages. As an example, in Figure 36, the DataSet TestRpt2 resides in the LLN0 of the logical device MyLD. This DataSet has two members—the references to the data objects Pos (MyLD/XCBR1.Pos) and BlkOpn (MyLD/XCBR1.BlkOpn). A change in the value of the data attribute stVal of data object Pos will be detected by an event monitor and cause the inclusion of the values of all data attributes of Pos in a message or log. (IEC 61850-7-1 2011: 40; IEC 61850-7-2 2010: 55.)

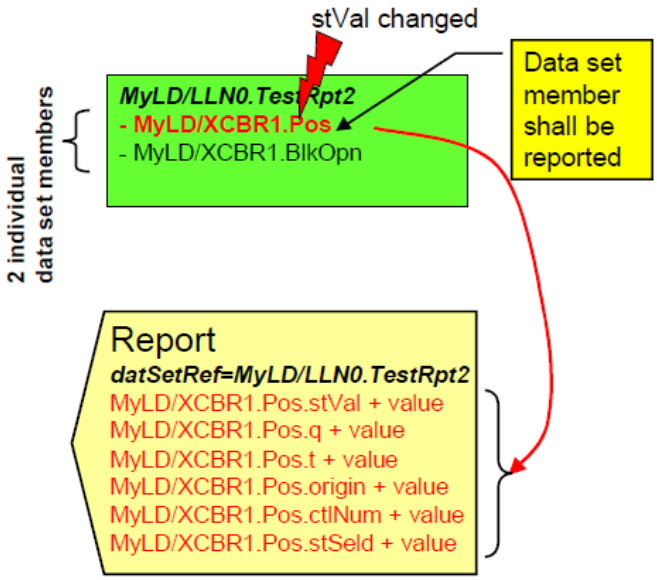


Figure 36. Example of a DataSet referenced by a control block. A value change in a data attribute of a data object referenced by the DataSet will be detected by an event monitor and cause a message to be generated and sent. (IEC 61850-7-1 2011: 41.)

In case of binary values, a false-to-true or true-to-false transition of the value of a data attribute referenced by a DataSet is detected by the event monitor. In case of analogue values, the event monitor may detect that the value of a data attribute deviates from a

specified deadband, or that the value crosses a specified range limit. The timestamp of a message is determined when an internal event occurs. It allows the subscribers to know that a status has changed and also the time of the last status change. (IEC 61850-7-1 2011: 37-39; IEC 61850-8-1 2010: 132.)

GOOSE messages are also sent with quality attributes that the subscribers can interpret. The quality attributes are used to mark invalid data and to detect failures in IEDs. As an example, if an IED sending GOOSE data has an oscillating input, the data from this input is marked as oscillating and the subscribing IEDs will not update the application due to invalid data. Instead of using the invalid data, a subscriber will use a preconfigured fail-safe value. (Hakala-Ranta, Rintamäki & Starck 2009.)

4.8. The generic substation event class model

The generic substation event (GSE) class model applies to the exchange of the values of a collection of objects referenced by a DataSet. It provides an efficient method for system-wide delivery of event information through the use of multicast services. The information exchange is based on a publisher/subscriber mechanism. (IEC 61850-7-2 2010: 131-133.)

An overview of the classes and services of the GOOSE communication model is given in Figure 37. The conceptual operation of GOOSE goes as follows: when the event monitor detects an internal event, the values of the objects which are referenced by the DataSet are written into the local transmission buffer and sent over the network. The subscribers can then read the received values from their reception buffers. (IEC 61850-7-2 2010: 131-132.)

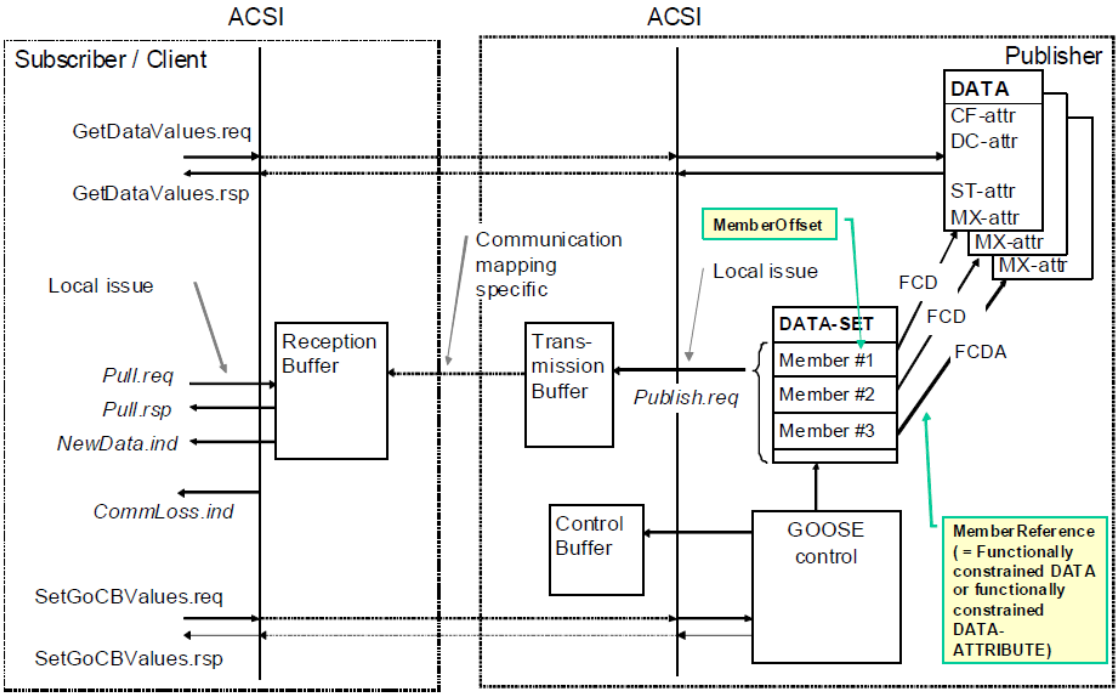


Figure 37. Conceptual GOOSE communication model. The members of the DataSet are assigned numbers, the MemberOffsets. When an internal event occurs, the values that the DataSet references are written into a local transmission buffer and transmitted over the network. The subscriber(s) may then retrieve the received values from the reception buffer. (IEC 61850-7-2 2010: 131.)

A GOOSE control block (GoCB) class is used to control the transmission of GOOSE messages. The SCSM for the GSE model maps the GoCB to the MMS GoCB. The services of the GoCB class are as follows (IEC 61850-7-2 2010: 131, 134-139; IEC 61850-8-1 2011: 87.):

- SendGOOSEMessage – service used by the GoCB to send a GOOSE message.
- GetGoReference – service used by a client to retrieve specific members of the DataSet.
- GetGOOSEElementNumber – service used by a client to retrieve the position of a selected member in the DataSet.

- GetGoCBValues – service used by a client to retrieve the attribute values of the GoCB.
- SetGoCBValues – service used by a client to set the attribute values of the GoCB.

An implementation declaring support for the SendGOOSEMessage, GetGoReference, and GetGOOSEElementNumber must use the GSE management and GOOSE communication profile, given in Appendix 1. (IEC 61850-8-1 2011: 29.)

When the GoCB needs to transmit a message, it uses the SendGOOSEMessage service. The SCSM for GOOSE messages encodes the values stored in the transmission buffer along with the name of the DataSet into a GOOSE message using the Abstract Syntax Notation One (ASN.1) Basic Encoding Rules (BER). SCSM-specific services[26] of the communication network are responsible to update the local reception buffers of the subscribers. The subscribers can then decode and read the received values from their reception buffers. (IEC 61850-7-2 2010: 131-132, 134; IEC 61850-8-1 2011: 93, 138.)

The SCSM used for GOOSE messages utilizes a retransmission scheme to achieve a sufficient level of reliability for the connectionless communication. This is illustrated in Figure 38. When an internal event occurs, a GOOSE message is immediately transmitted. The same message is then retransmitted repeatedly within the shortest retransmission time T1. After the first few rapid retransmissions, the retransmission time intervals increases gradually to T2 and T3 before finally settling at T0, the stable retransmission time interval. The retransmission continues at this rate until a new event occurs that sets the interval back to T1, and another burst of messages is transmitted. The retransmission scheme should be managed by a state machine that the publisher creates and maintains. The time delays between the retransmissions, the exact number of different retransmission intervals, and the total number intervals having a specific delay are not specified in the standard. (IEC 61850-8-1 2011: 93-94; Hou & Dolezilek 2008.)

[26] Defined by the Ethernet stack.

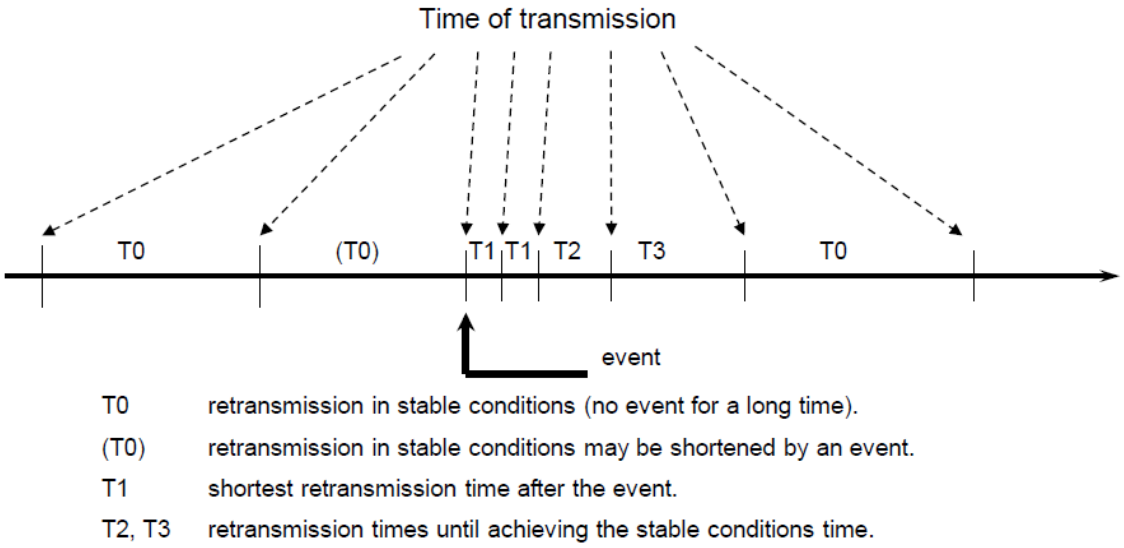


Figure 38. Retransmission intervals for the GOOSE message. When an internal event occurs, the GOOSE message is transmitted repeatedly between increasing time intervals, until finally settling at a stable time interval. (IEC 61850-8-1 2011: 93.)

The periodically retransmitted message under stable conditions can be seen as a "keep-alive" message. Each GOOSE message carries a user-specified timeAllowedToLive parameter that informs the subscriber of the maximum time to wait for the next retransmission. If the timeAllowedToLive is exceeded before a new message is received, the subscriber assumes that the association is lost—this is used to detect a failure of the publisher or a failure of the link between publisher and subscriber. (Wester & Adamiak 2012; IEC 61850-8-1 2011: 93.)

A device that is powered up sends information about its status as a GOOSE message. Any status changes are reported so that the receiving devices have knowledge of the time of the last status change. This knowledge allows devices to set local timers related to the event that caused the status change. All devices continue to resend their GOOSE messages over a long time cycle (the keep-alive interval) regardless of whether a status change has occurred or not, which ensures that the recently powered-up devices knows the status of their peer devices. (IEC 61850-7-2 2010: 132.)

The recommended multicast ranges for GOOSE and Sampled Values are given in Table 6. A multicast address is a 6 octet string. An un-configured multicast address has the value 00 for every octet. (IEC 61850-8-1 2011: 143.)

Table 6. Recommended multicast addresses for GOOSE, GSSE, and sampled values. (IEC 61850-8-1 2011: 143.)

Service	Recommended address range assignments	
	Starting address (hexadecimal)	Ending address (hexadecimal)
GOOSE	01-0C-CD-01-00-00	01-0C-CD-01-01-FF
GSSE	01-0C-CD-02-00-00	01-0C-CD-02-01-FF
Multicast sampled values	01-0C-CD-04-00-00	01-0C-CD-04-01-FF

Although switches perform multicast filtering and utilize VLANs, an IED still has to decode a received GOOSE message to see if it is a subscriber to that particular message. In order to achieve a sufficient rate of filtering and to prevent flooding of the IEDs that are not subscribing to a particular message, the decoding should be done by hardware—the SCSM for GOOSE performs multicast filtering at the DataLink layer. (Hou & Dolezilek 2008; IEC 61850-8-1 2011: 143; IEC 61850-7-1 2011: 34.)

The compatible logical node class LGOS is used for the monitoring of GOOSE messages. There shall be one instance of LGOS per GOOSE subscription for a given GOOSE source. It allows diagnosing the subscription state of a GOOSE message, and can contain optional information e.g. about the GoCB. The only mandatory data object in LGOS is St, status of the subscription (True = active, False = not active). (IEC 61850-7-4 2010: 25; IEC 61850-7-1 2011: 51.)

5. CONCLUSIONS

In this thesis, the international standard IEC 61850 was studied and a simple application based on an open source platform was developed to enable a Wärtsilä Local Display Unit 20 (LDU) to communicate with a Vamp 50 protection relay using the IEC 61850 Generic Object Oriented Substation Events (GOOSE) communication protocol.

The Rapid61850 platform has been shown to be a useful (although entirely free) asset for learning how the hierarchical data modelling and data flow modelling is done in IEC 61850. However, a few bugs and curiosities were found in the software—the bugs that were found and the settings that were done in Rapid61850 are listed in Appendix 6. The bugs have been reported to M.Sc. Steven Blair, creator of the software platform. Most of the bugs have already been removed from Rapid61850.

The results obtained from the software testing described in Chapter 3 confirm that we had succeeded in our objective of developing an application that enables the LDU to transmit and receive GOOSE messages. Although the created software cannot be used for real-life applications, its development has given valuable experience on IEC 61850.

5.1. Discussion

Many difficulties were encountered during the study of IEC 61850. The different parts of IEC 61850 were written by different groups of people from different countries, each of which left its own mark on the grammar, thus obstructing the interpretation of the standard. In general, IEC 61850 can be deemed as a standard hard to grasp. The text is, occasionally, very confusing to a reader who is not a domain expert, and many diagrams are more confusing than helpful. Additionally, the abstract nature of the ACSI has an obvious drawback in that it provides little information on how it should be implemented in practice.

Besides being the most informative part, IEC 61850-7-1 was also the most problematic part of the standard, due to its many ambiguities. The following statement can be found

on page 82: “Since many DATA classes use the same details (ATTRIBUTES), these details are therefore collected for re-use in common data classes (common to many DATA classes).” The statement is problematic for two reasons: Firstly, the word “ATTRIBUTES” might refer to both data attributes and to the attributes of a class model (i.e., not necessarily data attributes). The second problem is the reference to the “DATA” class, which does not exist in the second edition of IEC 61850-7-2, because the class name has apparently been changed from DATA to GenDataObjectClass.

The relations between the compatible data objects, the GenDataObjectClass, the common data classes, and the data attributes were not easily settled. The definitions are spread out all over IEC 61850-7-1. Fortunately (Liang & Campbell 2008) and (Mackiewicz 2006) provided their own viewpoints, which helped to clarify these relations.

IEC 61850-7-1 states on page 64 that “...LPHD must be defined in at least one logical device”. Presumably, it means that if a physical device contains several logical devices, at least one of them must contain an LPHD. IEC 61850-7-2 states on page 38 that “Each GenLogicalDeviceClass shall have one and only one logical-node-zero (LLN0) and it may have no or several other LogicalNodes.” Although there is no specific mention of the LPHD, it is clearly not mandatory in every logical device. (Liang & Campbell 2008) states the following: “Besides the regular logical nodes for functions, the standard also requires every logical device to have two specific logical nodes: Logical Node Zero (LN0) and LPHD, which correspond to the logical device and the physical device, alternatively.” This (incorrect) statement might be due to changes made between the first and the second edition of the standard. Most of the scientific articles that were studied refer to the first edition.

It is not entirely obvious what the TPAA and MCAA represent in practice, as only the MAC address and the Application ID for the GOOSE messages to be received needed to be configured to realize communication. (Blair 2013) suggests that the TPAA and MCAA are just conceptual associations which exist when the communication is

configured correctly, and that it is up to the SCSM to ensure that the requirements for either TPAA or MCAA are met.

What the SCSM corresponds to in Rapid61850 is also not obvious. (Blair 2013) suggests that the entire GOOSE packet generation done by the Rapid61850 library is the SCSM and that the GOOSE retransmission scheme is a part of it. The retransmission is implemented in the SCSM instead of the GoCB because platform-dependent timers are needed to realize the algorithm. (Blair 2013.)

I was not able to find a sensible definition as to what a "server" is. IEC 61850-6 states the following on page 24: "Typically, a switch controller LN may receive data as a client from a process bus, and provide data as a server to the inter-bay bus" which seem to suggest that the server is a logical node. However, the server is modelled as its own hierarchy level in the SCL, and many different sources state that it is the physical device that is defined as a server.

There seems to be some inconsistencies regarding the mapping of the GSE management profile in IEC 61850-8-1. Table 36 on page 58 tells that the GSE management profile shall implement the ACSI TPAA. However, tables 7 and 8 on pages 29 and 30, respectively, gives application profiles and transport profiles for GSE management/ GOOSE which bypass the Session, Transport and Network layers and thereby avoids any possibility of connection-oriented communication. I have assumed that the SCSM for a specific protocol implements the application profiles and transport profiles given for that protocol.

Some efforts have been made to reduce the ambiguities in IEC 61850, like the publishing of the IEC 61850-9-2LE guideline document which specifies the sampling rates of merging units. The specifications of IEC 61850 in general are still too loose to guarantee interoperability between IEDs from different manufacturers. Loosely defined and ambiguous specifications easily lead to a variety of proprietary versions of IEC 61850 that are not necessarily interoperable.

The technical issues (Tissues) website[27] is used to share information about the definition and use of IEC 61850. The site lists the technical issues found by various users and their proposals for improvements. Tissues are used as a basis for the next edition of the standard.

5.2. Future work

The developed software supports only a subset of the functionalities of IEC 61850, and more work is therefore required to make a module fully support the standard. However, the good results obtained from the software testing justify further research and development. Three different alternatives have been suggested for future work:

1. develop the Rapid61850 platform to support all the required functionalities,
2. replace Rapid61850 with a commercial communication stack that implements the required functionalities, or
3. develop a new communication stack from scratch.

Some of the missing functionalities in Rapid61850 are the trigger options for the GOOSE control block, a GOOSE retransmission scheme based on a state machine capable of real-time operation, validation of received messages, and the MMS services.

Some examples of vendors providing commercial communication stacks are:

- Triangle Microworks Inc.[28] provides IEC 61850 source code libraries written in C. An IEC 61850 Test Suite is also available.
- INFO TECH[29] provides an IEC 61850 software library written in C.

---

[27] http://www.tissues.iec61850.com/parts.mspx

[28] http://www.trianglemicroworks.com/documents/IEC_61850_Source_Code_Libraries.pdf

[29] http://www.infotech.pl/

Regardless of which path that will be followed in future work, the SCL language should be studied more thoroughly than what was done in this thesis. Definitions of all SCL elements and attributes are available in IEC 61850-6, but these were ignored. The development of the SCD file mainly followed the “trial and error” method.

An important issue that needs to be considered when implementing IEC 61850 for commercial applications is the conformance to part 10 of the standard. IEC 61850-10 ‘Conformance testing’ specifies (IEC 61850-10 2005: 6-7):

- standard techniques for testing of conformance of implementations,
- abstract test cases for conformance testing of devices,
- specific measurement techniques to be applied when declaring performance parameters, and
- the metrics to be measured within devices according to the requirements defined in IEC 61850-5.

Information on tests regarding EMC requirements and environmental conditions can be found in IEC 61850-3. (IEC 61850-10 2005: 6.)

(Vähämäki 2013) states that an official conformance test is currently not required—a product can be advertised as being IEC 61850 conformant even if no official conformance test has been performed on it. Customers may however require that the vendor supplies a document called a *Conformance Statement* which provides the IEC 61850 conformance details of a product, i.e., it lists the aspects of the standard that are supported. The Conformance Statement consists of the following sections (VAMP 2011):

- The Protocol Implementation Conformance Statement (PICS) lists the ACSI conformance statements, i.e., what aspects of the ACSI that are supported by the product, and the protocol profile support, i.e., what application profiles and transport profiles are supported.
- The Model Implementation Conformance Statement (MICS) lists the application functions supported by the server, i.e., the supported logical nodes classes from

IEC 61850-7-4. It also lists supported logical node type extensions, common data class type extensions, and enumerated type extensions.

- The Protocol Implementation eXtra Information for Testing (PIXIT)[30] lists the extra information on the IEC 61850 server implementation, for example the retransmission intervals, the maximum number of elements in GOOSE publisher DataSets, and the trigger options.
- The Tissue Implementation Conformance Statement (TICS) lists the implemented technical issues corrections found on the Tissues website.

The PICS, MICS, PIXIT and TICS might have to be delivered as separate documents on request of the customer. (Vähämäki 2013.)

An official conformance test[31] can be performed on an individual product by an independent test laboratory. The testing is done according to the IEC 61850-10 and the Conformance Statement of the product. A test lab with a certified ISO 9000[32] or ISO 17025 quality system can award a product a Level A certificate after the product has successfully passed a series of tests. The test is based on the UCA International Users Group Device Test Procedures version 2.2b. The Level A certificate proves that the product has not shown to be non-conforming to those parts of the IEC 61850 standard that the product is claimed to conform to in its Conformance Statement. (KEMA 2011.)

---

[30] Contains system specific information regarding the capabilities of the system to be tested which are outside the scope of the IEC 61850 series. Provides information regarding the physical setup that is not part of the ACSI. This could be information regarding the hardware, socket, and other information. The PIXIT shall not be subjected to standardization. (IEC 61850-2 2003: 20.)

[31] Check of data flow on communication channels in accordance with the standard conditions concerning access organization, formats and bit sequences, time synchronization, timing, signal form and level and reaction to errors. (IEC 61850-2 2003: 7.)

[32] The conformance test should be carried out by an ISO 9001 certified organization or system integrator. (IEC 61850-2 2003: 7.)

REFERENCES

Aguilar, Rene & James Ariza (2010). *Testing and configuration of IEC 61850 multivendor protection schemes.* IEEE PES Transmission and Distribution Conference and Exposition, New Orleans, LA, USA.

Antonova, Galina, Lars Frisk & Jean-Charles Tournier (2011). *Communication redundancy for substation automation.* 64th Annual Conference for Protective Relay Engineers.

Apostolov, Alexander P. (2010). *UML and XML use in IEC 61850.* IEEE PES Transmission and Distribution Conference and Exposition 2010.

Bekker, Dorran D., Peter Diamandis & Tim Tibbals (2010). *IEC 61850 – More than just GOOSE: a case study of modernizing substations in Namibia* [online]. Proceedings of the 12th Annual Western Power Delivery Automation Conference, Spokane, WA, April 2010 [cited 11.07.2012]. Available from World Wide Web: <URL: https://www.selinc.com/ WorkArea/DownloadAsset.aspx?id=7399>.

Blair, Steven M. (2012). *Rapid-prototyping protection schemes with IEC 61850* [online]. README.md file for Rapid61850 [cited 25.12.2012]. Available from World Wide Web: <URL: https://github.com/stevenblair/rapid61850>.

Blair, Steven M. (2013). E-mail conversation with M.Sc. Steven Blair, creator of the Rapid61850 platform, University of Strathclyde, Glasgow, UK, 30.1.2013.

Blair, Steven, Campbell Booth & Graeme Burt (2011a). *Architecture for automatically generating an efficient IEC 61850-based communications platform for the rapid prototyping of protection schemes* [online]. Research paper [cited 2.11.2012]. Available from World Wide Web: <URL: http://personal.strath.ac.uk/ steven.m.blair/OP054.pdf>.

Blair, Steven M., Campbell Booth & Graeme Burt (2011b). *Rapid-prototyping protection schemes with IEC 61850* [online]. Presentation slides [cited 2.11.2012]. Available from World Wide Web: <URL: http://personal.strath.ac.uk/steven.m.blair/Steven%20M.%20Blair%20-%20PAC%20World%202011%20Presentation.pdf>.

Blair, Steven M., Frederico Coffele, Campbell D. Booth & Graeme M. Burt (2012). *An open platform for rapid-prototyping of protection and control schemes with IEC 61850* [online] IEEE transactions on power delivery 11/2012; 1 [cited 25.12.2012]. Available from World Wide Web: <URL: http://personal.strath.ac.uk/steven.m.blair/S-Blair-Rapid-IEC-61850-preprint.pdf.

Dawidczak, Henry & Heiko Englert (2010). *IEC 61850 interoperability and use of flexible object modeling and naming* [online]. Siemens AG, Energy Automation Nuremberg, Germany [cited 11.7.2012]. Available from World Wide Web: < http://romvchvlcomm.pbworks.com/f/ Paper+Dawidczak_Englert_RZA2010.pdf >.

De Mesmaeker, Ivan, Peter Rietmann, Klaus-Peter Brand & Petra Reinhardt (2005). *Substation automation based on IEC 61850*. Cigré SC B5 6th Regional CIGRÉ conference in Cairo, November 21-23, 2005.

Garcia, Luis Martin (2008). *Programming with Libpcap – Sniffing the network from our own application* [online]. Haking magazine, 2/2008 issue, vol 3 [cited 15.1.2013]. Available from World Wide Web: <URL: http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf >.

Goldberg, Kevin Howard (2009). *Visual Quickstart Guide – XML, Second Edition.* USA: Peachpit Press. ISBN 978-0-321-55967-8.

Hakala-Ranta, Antti, Olli Rintamäki & Janne Starck (2009). *Utilizing possibilities of IEC 61850 and GOOSE*. 20th International Conference on Electricity Distribution, Prague 8-11 June 2009.

Hou, Daqing & David Dolezilek (2008). *IEC 61850 – What it can and cannot offer to traditional protection schemes* [online]. Schweitzer Engineering Laboratories, Inc. [cited 26.7.2012]. Available from World Wide Web: <URL: http:// http://www2.selinc.com/techpprs/6335_IEC61850_DH-DD_20080912.pdf>.

IEC 61850-1 (2003). *Communication networks and systems in substations – Part 1: Introduction and overview,* 1st edition.

IEC 61850-2 (2003). *Communication networks and systems in substations – Part 2: Glossary,* 1st edition.

IEC 61850-5 (2003). *Communication networks and systems in substations – Part 5: Communication requirements for functions and device models,* 1st edition.

IEC 61850-6 (2009). *Communication networks and systems for power utility automation – Part 6: Configuration description language for communication in electrical substations related to IEDs,* 2nd edition.

IEC 61850-7-1 (2011). *Communication networks and systems for power utility automation – Part 7-1: Basic communication structure – Principles and models,* 2nd edition.

IEC 61850-7-2 (2010). *Communication networks and systems for power utility automation – Part 7-2: Basic information and communication structure – Abstract communication service interface (ACSI),* 2nd edition.

IEC 61850-7-3 (2010). *Communication networks and systems for power utility automation – Part 7-3: Basic communication structure – Common data classes,* 2nd edition.

IEC 61850-7-4 (2010). *Communication networks and systems for power utility automation – Part 7-4: Basic communication structure – Compatible logical node classes and data object classes,* 2nd edition.

IEC 61850-7-420 (2009). *Communication networks for power utility automation – Part 7-420: Basic communication structure – Distributed energy resources logical nodes*, 1st edition.

IEC 61850-8-1 (2011). *Communication networks and systems for power utility automation – Part 8-1: Specific communication service mapping (SCSM) – Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3*, 2nd edition.

IEC 61850-9-1 (2003). *Communication networks and systems in substations – Part 9-1: Specific Communication Service Mapping (SCSM) – Sampled values over serial unidirectional multidrop point to point link*, 1st edition.

IEC 61850-9-2 (2011). *Communication networks and systems for power utility automation – Part 9-2: Specific communication service mapping (SCSM) – Sampled values over ISO/IEC 8802-3*, 2nd edition.

IEC 61850-10 (2005). *Communication networks and systems in substations – Part 10: Conformance testing*, 1st edition.

Ingram, David M. E., Pascal Schaub & Duncan A. Campbell (2011). *Multicast traffic filtering for sampled value process bus networks.* IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society.

Ingram, David M. E., Pascal Schaub & Duncan A. Campbell (2012). *Use of precision time protocol to synchronize sampled-value process buses.* IEEE transactions on instrumentation and measurement, vol. 61, no. 5, May 2012.

Janssen, Marco C. & Alexander Apostolov (2008). *IEC 61850 impact on substation design.* IEEE/PES Transmission and Distribution Conference and Exposition 2008.

KEMA (2011). *IEC 61850 Certificate Level A for the product VAMP 50.*

Kerrisk, Michael (2010). *The Linux Programming Interface – A Linux and UNIX System Programming Handbook.* USA: No Starch Press. ISBN: 978-1-59327-220-3.

Kirrmann, Hubert, Peter Rietmann & Steven Kunsman (2008). *Standard network redundancy using IEC 62439*. PAC World Magazine, Autumn 2008 Issue.

Liang, Yingyi & Roy H. Campbell (2008). *Understanding and simulating the IEC 61850 standard* [online]. Technical report [cited 11.7.2012]. Available from World Wide Web: <URL: http://hdl.handle.net/2142/11457>.

Mackiewicz, Ralph E. (2006). *Overview of IEC 61850 and benefits*. IEEE Power Engineering Society General Meeting, 2006.

McGhee, Jim & Maciej Goraj (2010). *Smart high voltage substation based on IEC 61850 process bus and IEEE 1588 time synchronization.* 2010 1st IEEE International Conference on Smart Grid Communications.

Mohagheghi, Salman, Jean-Charles Tournier, James Stoupis, Laurent Guise, Thierry Coste, Claus A. Andersen & Jakob Dall (2011). *Applications of IEC 61850 in Distribution Automation.* 2011 IEEE PES Power Systems Conference & Exposition. Phoenix, Arizona, USA.

Moore, Roger & Maciej Goraj (2011). *New paradigm of smart transmission substation – practical experience with Ethernet based fiber optic switchyard at 500 kilovolts*. 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe).

Moore, Roger, Rene Midence & Maciej Goraj (2010). *Practical experience with IEEE 1588 high precision time synchronization in electrical substation based on IEC 61850 process bus.* Power and Energy Society General Meeting, 2010 IEEE.

Ozansoy, Cagil Ramadan (2006). *Design & Implementation of a Universal Communications Processor for Substation Integration, Automation and Protection* [online]. PhD Thesis, Victoria University, Australia [cited 23.1.2013]. Available from World Wide Web: <URL: http://vuir.vu.edu.au/527/>.

Scaglia, Sergio (2007). *The Embedded Internet*. Great Britain: Pearson Education Limited. ISBN 978-0-321-30638-8.

Schnakofsky, Alejandro (2011). *Digitizing copper – what to consider for a successful integration* [online]. Presentation slides [cited 2.8.2012]. Available from World Wide Web: <URL: http://www05.abb.com/global/scot/scot299.nsf/veritydisplay/df10955eea73395785257848006cd255/$file/digitizing%20copper%20webinar%20rev2.pdf>.

Schwarz, Karlheinz (2008). *Manufacturing message specification – ISO9506 (MMS)* [online]. Schwarz Consulting Company, Karlsruhe, Germany [cited 19.12.2012]. Available from World Wide Web: <URL: http://www.nettedautomation.com/download/MMS-R1-2_2008-02-26.pdf>.

SearchNetworking (2013). *Definition of graceful degradation* [online]. [Cited 14.2.2013]. Available from World Wide Web: <URL: http://searchnetworking.techtarget.com/definition/graceful-degradation>.

Starck, Janne (2012). E-mail conversation with M.Sc. Janne Starck, Product manager at ABB Finland, 26.11.2012.

Stevens, W. Richard (1998). *UNIX Network Programming – Networking APIs: Sockets and XTI Volume 1 2nd edition.* USA: Prentice Hall. ISBN 0-13-490012-X.

Ubuntu Manuals (2013). *AF_PACKET* [online]. Ubuntu Manpage Repository [cited 23.1.2013]. Available from World Wide Web: <URL: http://manpages.ubuntu.com/manpages/jaunty/man7/packet.7.html>.

VAMP (2009). *IEC 61850 interface configuration for VAMP 50,51,51,257,259* [online] Configuration manual AN61850.EN001 [cited 8.11.2012]. Available from World Wide Web: <URL: http://www.vamp.fi/In%20English/Home/Default.aspx>.

VAMP (2010a). *IEC 61850 interface configuration for VAMP relays* [online]. Configuration manual AN61850.EN002 [cited 8.11.2012]. Available from World Wide Web: <URL: http://www.vamp.fi/In%20English/Home/Default.aspx>.

VAMP (2010b). *IEC 61850 GOOSE interface in VAMP relays configuration instruction* [online]. Configuration manual AN61850.EN003 [cited 8.11.2012]. Available from World Wide Web: <URL: http://www.vamp.fi/In%20English/Home/Default.aspx >.

VAMP (2011). *VAMP Embedded IEC 61850 Server Conformance Statement for VAMP 50, 51, 52, 55, 59, 210, 230, 245, 255, 260, 265, 265M, 257, 259, 321.*

Vähämäki, Olavi (2013). E-mail conversation with M.Sc. Olavi Vähämäki, R&D Director at Vamp Ltd., 31.1.2013.

Wester, Craig & Mark Adamiak (2012). *Practical applications of peer-to-peer messaging in industrial facilities.* 2012 65th Annual Conference for Protective Relay Engineers.

Wikipedia (2013). *pcap* [online]. [Cited 15.1.2013]. Available from World Wide Web: <URL: http://en.wikipedia.org/wiki/Pcap>.

Xyngi, I. & M. Popov (2010). *IEC 61850 overview – where protection meets communication.* The 10th IET International Conference on Developments in Power System Protection. Manchester, UK.

Yashwant K. & K. Shanti Swarup (2011). *Modeling an IEC61850 based substation automation system.* India Conference (INDICON), 2011 Annual IEEE.

Yongli, Zhu, Wang Dewen, Wang Yan & Zhao Wenqing (2009). *Study on interoperable exchange of IEC 61850 data model*. 4th IEEE Conference on Industrial Electronics and Applications.

Yun, Pan (2011). *Research on IED configurator based on IEC 61850.* 2011 International Conference on Control, Automation and Systems Engineering (CASE).

APPENDICES

APPENDIX 1. Application profiles and transport profiles

The application profile and transport profile used for client/server communication in IEC 61850 are given in Table 7 and Table 8, respectively. There are 2 transport profiles that may be used by the client/server application profile: TCP/IP or OSI. An implementation that claims conformance to IEC 61850 shall implement the TCP/IP profile as a minimum. (IEC 61850-8-1 2011: 26.)

Table 7. Services and protocols for client/server communication application profile. (IEC 61850-8-1 2011: 26.)

OSI model layer	Specification			m/o
	Name	Service specification	Protocol specification	
Application	Manufacturing Message Specification	ISO 9506-1:2003	ISO 9506-2:2003	m
	Association Control Service Element	ISO/IEC 8649:1996	ISO/IEC 8650-1:1996	m
Presentation	Connection Oriented Presentation	ISO/IEC 8822:1994	ISO/IEC 8823-1:1994	m
	Abstract Syntax	ISO/IEC 8824-1:2008	ISO/IEC 8825-1:2008	m
Session	Connection Oriented Session	ISO/IEC 8326:1996	ISO/IEC 8327-1:1997	m

Table 8. Services and protocols for client/server TCP/IP transport profile. (IEC 61850-8-1 2011: 27.)

OSI Model Layer	Specification			m/o
	Name	Service specification	Protocol specification	
Communication	Requirement for internet host	RFC 1122		m
Transport	ISO Transport on top of TCP	RFC 1006		m
	Internet Control Message Protocol (ICMP)	RFC 792		m
	Transmission Control Protocol (TCP)	RFC 793		m
Network	Internet Protocol	RFC 791		m
	An Ethernet Address Resolution Protocol (ARP)	RFC 826		m
Link Redundancy	Parallel Redundancy Protocol and High Availability Seamless Ring	IEC 62439-3 – PRP1 or HSR		o
	Rapid Spanning Tree Protocol (RSTP)	IEEE 802.1D		o
DataLink	Standard for the transmission of IP datagrams over Ethernet networks	RFC 894		m
	Carrier Sense Multiple Access with collision detection (CSMA/CD)	ISO/IEC 8802-3:2000		m
Physical (option 1)	10Base-T/100Base-T	ISO/IEC 8802-3:2000		c1
	Interface connector and contact assignments for ISDN Basic Access Interface. [a]	ISO/IEC 8877:1992		
Physical (option 2)	Fibre optic transmission system 100Base-FX	ISO/IEC 8802-3:2000		c1
	Fiber optic transmission system 1000Base-LX	ISO/IEC 8802-2:1998, ISO/IEC 8802-3:2000		
	Basic Optical Fibre Connector. [b]	IEC 60874-10-1, IEC 60874-10-2 and IEC 60874-10-3		
	Basic Optical Fibre Connector. [c]	IEC 61754-22		

The application profile and transport profile for GSE Management/GOOSE communication are given in Table 9 and Table 10, respectively. The GSE Management/GOOSE communication profile shall be used for any implementation claiming conformance to IEC 61850 and declaring support for one of the services given in Table 11. (IEC 61850-8-1 2011: 29.)

Table 9. Service and protocols for GSE Management and GOOSE communication application profile. (IEC 61850-8-1 2011: 29.)

OSI model layer	Specification			m/o
	Name	Service specification	Protocol specification	
Application	GSE/GOOSE protocol	See Annex A		m
Presentation	Abstract Syntax	NULL		m
Session				

Table 10. Service and protocols for GSE Management and GOOSE communication transport profile (IEC 61850-8-1 2011: 30.)

OSI model layer	Specification			m/o
	Name	Service specification	Protocol specification	
Transport				
Network				
Link Redundancy	Parallel Redundancy Protocol and High Availability Seamless Ring	IEC 62439-3 – PRP1 or HSR		o
	Rapid Spanning Tree Protocol (RSTP)	IEEE 802.1D		o
DataLink	Priority Tagging/ VLAN	IEEE 802.1Q		m
	Carrier Sense Multiple Access with collision detection (CSMA/CD).	ISO/IEC 8802-3:2000		m
Physical (option 1)	10Base-T/100Base-T	ISO/IEC 8802-3:2000		c
	Interface connector and contact assignments for ISDN Basic Access Interface. [a]	ISO/IEC 8877:1992		
Physical (option 2)	Fibre optic transmission system 1000Base-LX	ISO/IEC 8802-2:1998, ISO/IEC 8802-3:2000		c
	Basic Optical Fibre Connector. [b]	IEC 60874-10-1, IEC 60874-10-2 and IEC 60874-10-3		

Table 11. Services requiring GSE Management and GOOSE communication profile. (IEC 61850-8-1 2011: 29.)

Model	IEC 61850-7-2 service
Generic Substation Event	GetGoReference GetGOOSEElementNumber SendGOOSEMessage

APPENDIX 2. The SCD file used in the GOOSE software project

```
<?xml version="1.0" encoding="utf-8"?>
<SCL xmlns="http://www.iec.ch/61850/2006/SCL" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.iec.ch/61850/2006/SCL">
  <Header id="SCL Example T1-1" nameStructure="IEDName" />
  <Communication>
    <SubNetwork name="W01" type="8-MMS">
      <Text>Station bus</Text>
      <BitRate unit="b/s">10</BitRate>
      <ConnectedAP iedName="Laptop" apName="S1">
        <Address>
          <P type="IP">10.4.128.104</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">10.4.128.253</P>
          <P type="OSI-TSEL">00000001</P>
          <P type="OSI-PSEL">01</P>
          <P type="OSI-SSEL">01</P>
        </Address>
        <GSE ldInst="C1" cbName="Status">
          <Address>
            <P type="MAC-Address">01-0C-CD-01-00-05</P>
            <P type="APPID">3002</P>
            <P type="VLAN-PRIORITY">4</P>
          </Address>
        </GSE>
        </ConnectedAP>
        <ConnectedAP iedName="VAMP" apName="P1">
        <Address>
            <P type="OSI-PSEL">00000001</P>
            <P type="OSI-SSEL">0001</P>
            <P type="OSI-TSEL">0001</P>
            <P type="IP">10.4.128.109</P>
            <P type="IP-SUBNET">255.255.255.0</P>
            <P type="IP-GATEWAY">10.4.128.254</P>
        </Address>
        <GSE ldInst="Relay" cbName="gcb1">
        <Address>
            <P type="VLAN-PRIORITY">4</P>
            <P type="MAC-Address">01-0C-CD-01-00-00</P>
            <P type="APPID">1091</P>
        </Address>
        </GSE>
        </ConnectedAP>
    </SubNetwork>
  </Communication>
```

```
<!-- IED Laptop was manually added -->
<IED name="Laptop">
  <AccessPoint name="S1">
    <Server>
      <Authentication none="true"/>
      <LDevice inst="C1">
        <LN0 lnType="LN0" lnClass="LLN0" inst="">
          <DataSet name="Status">
            <FCDA ldInst="C1" prefix="" lnInst="1" lnClass="DCIP"
            doName="EngOnOff" daName="stVal" fc="ST" />
          </DataSet>
          <GSEControl name="Status" datSet="Status" appID="Status"
          confRev="1" />
        </LN0>
        <LN lnType="DCIPa" lnClass="DCIP" inst="1" >
          <Inputs>
            <ExtRef iedName="VAMP" ldInst="Relay" prefix="VI1"
            lnClass="GGIO" lnInst="137" doName="SPCSO" daName="stVal" />
            <ExtRef iedName="VAMP" ldInst="Relay" prefix="VI2"
            lnClass="GGIO" lnInst="138" doName="SPCSO" daName="stVal" />
          </Inputs>
        </LN>
      </LDevice>
    </Server>
  </AccessPoint>
</IED>
<!-- IED VAMP was generated by Vampset (name="TEMPLATE") and manually
edited for convenience -->
<IED name="VAMP" type="VAMP 50">
  <AccessPoint name="P1">
    <Server>
      <Authentication none="true"/>
      <LDevice inst="Relay">
        <LN0 lnType="LLN0_0" lnClass="LLN0" inst="">
          <DataSet name="DSG1">
            <FCDA ldInst="Relay" prefix="VI1" lnClass="GGIO" lnInst="137"
            doName="SPCSO" daName="stVal" fc="ST"/>
            <FCDA ldInst="Relay" prefix="VI2" lnClass="GGIO" lnInst="138"
            doName="SPCSO" daName="stVal" fc="ST"/>
          </DataSet>
          <Inputs>
            <ExtRef intAddr="NI1" iedName="Laptop" ldInst="C1" prefix=""
            lnClass="DCIP" lnInst="1" doName="EngOnOff" daName="stVal"/>
          </Inputs>
          <GSEControl name="gcb1" type="GOOSE" appID="VAMP" confRev="1"
          datSet="DSG1"/>
        </LN0>
```

```
        <LN lnType="GGIO_4" lnClass="GGIO" prefix="VI1" inst="137">
          <DOI name="SPCSO">
            <DAI name="ctlModel">
              <Val>direct-with-normal-security</Val>
            </DAI>
          </DOI>
        </LN>
        <LN lnType="GGIO_4" lnClass="GGIO" prefix="VI2" inst="138">
          <DOI name="SPCSO">
            <DAI name="ctlModel">
              <Val>direct-with-normal-security</Val>
            </DAI>
          </DOI>
        </LN>
      </LDevice>
    </Server>
  </AccessPoint>
</IED>
<DataTypeTemplates>
  <!-- Templates for IED Laptop (manually added) -->
  <LNodeType id="LN0" lnClass="LLN0"/>
  <LNodeType id="DCIPa" lnClass="DCIP">
    <DO name="EngOnOff" type="simpleSPS"/>
    <DO name="EngRPM" type="simpleMV"/>
  </LNodeType>
  <DOType id="simpleSPS" cdc="SPS">
    <DA name="stVal" fc="ST" bType="BOOLEAN" dchg="true"/>
    <DA name="q" fc="ST" bType="Quality" qchg="true"/>
    <DA name="t" fc="ST" bType="Timestamp"/>
  </DOType>
  <!-- Templates for IED VAMP (generated by Vampset) -->
  <LNodeType id="GGIO_4" lnClass="GGIO">
    <DO name="Mod" type="INC_0"/>
    <DO name="Beh" type="INS_0"/>
    <DO name="Health" type="INS_1"/>
    <DO name="NamPlt" type="LPL_0"/>
    <DO name="SPCSO" type="SPC_1"/>
  </LNodeType>
  <LNodeType id="LLN0_0" lnClass="LLN0">
    <DO name="Mod" type="INC_0"/>
    <DO name="Beh" type="INS_0"/>
    <DO name="Health" type="INS_1"/>
    <DO name="NamPlt" type="LPL_1"/>
    <DO name="Loc" type="SPS_0"/>
  </LNodeType>
  <LNodeType id="LPHD_0" lnClass="LPHD">
    <DO name="PhyNam" type="DPL_0"/>
    <DO name="PhyHealth" type="INS_1"/>
    <DO name="Proxy" type="SPS_0"/>
  </LNodeType>
  <DOType id="INC_0" cdc="INC">
    <DA name="stVal" fc="ST" bType="Enum" type="Mod"/>
```

```
  <DA name="q" fc="ST" bType="Quality"/>
  <DA name="t" fc="ST" bType="Timestamp"/>
  <DA name="ctlModel" fc="CF" bType="Enum" type="ctlModel"/>
</DOType>
<DOType id="INS_0" cdc="INS">
  <DA name="stVal" fc="ST" bType="Enum" type="Beh"/>
  <DA name="q" fc="ST" bType="Quality"/>
  <DA name="t" fc="ST" bType="Timestamp"/>
</DOType>
<DOType id="INS_1" cdc="INS">
  <DA name="stVal" fc="ST" bType="Enum" type="Health"/>
  <DA name="q" fc="ST" bType="Quality"/>
  <DA name="t" fc="ST" bType="Timestamp"/>
</DOType>
<DOType id="LPL_0" cdc="LPL">
  <DA name="vendor" fc="DC" bType="VisString255"/>
  <DA name="swRev" fc="DC" bType="VisString255"/>
  <DA name="d" fc="DC" bType="VisString255"/>
</DOType>
<DOType id="SPC_1" cdc="SPC">
  <DA name="stVal" fc="ST" bType="BOOLEAN"/>
  <DA name="q" fc="ST" bType="Quality"/>
  <DA name="t" fc="ST" bType="Timestamp"/>
  <DA name="ctlModel" fc="CF" bType="Enum" type="ctlModel"/>
</DOType>
<DOType id="LPL_1" cdc="LPL">
  <DA name="vendor" fc="DC" bType="VisString255"/>
  <DA name="swRev" fc="DC" bType="VisString255"/>
  <DA name="d" fc="DC" bType="VisString255"/>
  <DA name="configRev" fc="DC" bType="VisString255"/>
  <DA name="ldNs" fc="EX" bType="VisString255"/>
</DOType>
<DOType id="SPS_0" cdc="SPS">
  <DA name="stVal" fc="ST" bType="BOOLEAN"/>
  <DA name="q" fc="ST" bType="Quality"/>
  <DA name="t" fc="ST" bType="Timestamp"/>
</DOType>
<DOType id="DPL_0" cdc="DPL">
  <DA name="vendor" fc="DC" bType="VisString255"/>
</DOType>
<EnumType id="ctlModel">
  <EnumVal ord="0">status-only</EnumVal>
  <EnumVal ord="1">direct-with-normal-security</EnumVal>
  <EnumVal ord="2">sbo-with-normal-security</EnumVal>
  <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
  <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
</EnumType>
<EnumType id="orCategory">
  <EnumVal ord="0">not-supported</EnumVal>
  <EnumVal ord="1">bay-control</EnumVal>
  <EnumVal ord="2">station-control</EnumVal>
  <EnumVal ord="3">remote-control</EnumVal>
```

```
      <EnumVal ord="4">automatic-bay</EnumVal>
      <EnumVal ord="5">automatic-station</EnumVal>
      <EnumVal ord="6">automatic-remote</EnumVal>
      <EnumVal ord="7">maintenance</EnumVal>
      <EnumVal ord="8">process</EnumVal>
    </EnumType>
    <EnumType id="Beh">
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
    <EnumType id="Mod">
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
    <EnumType id="Health">
      <EnumVal ord="1">Ok</EnumVal>
      <EnumVal ord="2">Warning</EnumVal>
      <EnumVal ord="3">Alarm</EnumVal>
    </EnumType>
    <EnumType id="Check">
      <EnumVal ord="0">no-check</EnumVal>
      <EnumVal ord="1">synchrocheck</EnumVal>
      <EnumVal ord="2">interlocking-check</EnumVal>
      <EnumVal ord="3">both</EnumVal>
    </EnumType>
  </DataTypeTemplates>
</SCL>
```

APPENDIX 3. The main.c file used in the GOOSE software project

```
/**
 * Rapid-prototyping protection schemes with IEC 61850
 *
 * Copyright (c) 2012 Steven Blair
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
USA.
 */

#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
//#include <net/if.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include "iec61850.h"

#define BUFFER_LENGTH   2048
#define DEFAULT_IF "eth0"

int s;
int len = 0;
unsigned char buf[BUFFER_LENGTH] = {0};

int intervalselector = 0;
int ticks = 0;

int send_result = -1;
struct sockaddr_ll socket_address;
```

```
int createSocket(struct sockaddr_ll *sockaddr) {
    int sock;
    struct ifreq if_idx;
    struct ifreq if_mac;
    char ifName[IFNAMSIZ];

    sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (s == -1) {
        fprintf(stderr, "Unable to open the socket");
                exit(2);
    }

    fcntl(sock, F_SETFL, O_NONBLOCK);

    strcpy(ifName, DEFAULT_IF);
    /* Get the index of the interface to send on */
    memset(&if_idx, 0, sizeof(struct ifreq));
    strncpy(if_idx.ifr_name, ifName, IFNAMSIZ-1);
    if (ioctl(sock, SIOCGIFINDEX, &if_idx) < 0){
        fprintf(stderr, "SIOCGIFINDEX error");
                exit(2);
    }
    /* Get the MAC address of the interface to send on */
    memset(&if_mac, 0, sizeof(struct ifreq));
    strncpy(if_mac.ifr_name, ifName, IFNAMSIZ-1);
    if (ioctl(sock, SIOCGIFHWADDR, &if_mac) < 0){
        fprintf(stderr, "SIOCGIFHWADDR error");
                exit(2);
    }

    sockaddr->sll_ifindex = if_idx.ifr_ifindex;

    /* Get the current flags that the device might have */
    if (ioctl (sock, SIOCGIFFLAGS, &if_idx) == -1)
    {
     perror ("Error: Could not retrieve the flags from the device.\n");
     exit (1);
    }

    /* Set the old flags plus the IFF_PROMISC flag */
    if_idx.ifr_flags |= IFF_PROMISC;
    if (ioctl (sock, SIOCSIFFLAGS, &if_idx) == -1)
    {
      perror ("Error: Could not set flag IFF_PROMISC");
      exit (1);
    }

    sockaddr->sll_halen = ETH_ALEN;
    sockaddr->sll_addr[6]  = 0x00;/*not used*/
    sockaddr->sll_addr[7]  = 0x00;/*not used*/
    return sock;
}
```

```
void GSEcallbackFunction(CTYPE_INT32U timeAllowedToLive, CTYPE_TIMESTAMP T,
        CTYPE_INT32U stNum, CTYPE_INT32U sqNum)
{
    printf("Button F1: %s, timeAllowedToLive: %i\n",
    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.Relay_VI1GGIO_137
    _SPCSO_stVal == 1 ? "On" : "Off",timeAllowedToLive);

    if( Laptop.S1.C1.DCIPa_1.EngOnOff.stVal !=
    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.Relay_VI1GGIO_137_
    SPCSO_stVal )
    {
        Laptop.S1.C1.DCIPa_1.EngOnOff.stVal =
        Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.VAMP_Relay_DSG1.
        Relay_VI1GGIO_137_SPCSO_stVal;

        intervalselector = 0;
        ticks = 0;

        len = Laptop.S1.C1.LN0.Status.send(buf, 1, 1024);

        strncpy(socket_address.sll_addr,
        Laptop.S1.C1.LN0.Status.ethHeaderData.destMACAddress, ETH_ALEN);

        send_result = sendto(s, buf, len, 0,
        (struct sockaddr*)&socket_address, sizeof(socket_address));

        if (send_result == -1) {
            fprintf(stderr, "Send failed");
        }

        printf("Package sent, timeAllowedToLive: 1024\n");
    }
}
```

```
int main() {

    initialise_iec61850();
    s = createSocket(&socket_address);
    int i = 0;

    Laptop.S1.C1.DCIPa_1.gse_inputs_gcb1.datasetDecodeDone =
    &GSEcallbackFunction; // prototype

    int milliintervals[] = {1024,   1024,   4098,   16384,  32768};
    int ticksintervals[] = {0,      0,      4,      20,     40   };

    for(;; ticks++)
    {
        // Receive
        do {
            len = recvfrom(s, buf, BUFFER_LENGTH, 0, NULL, NULL);
            if(len){
                gse_sv_packet_filter(buf, len);
            }
        } while(len >= 0);

        if( ticks >= ticksintervals[intervalselector] )
        {
            //send old
            if( intervalselector < 4 )
            {
                intervalselector++;
            }

            len = Laptop.S1.C1.LN0.Status.send(buf, 0,
            milliintervals[intervalselector]);

            strncpy(socket_address.sll_addr,
            Laptop.S1.C1.LN0.Status.ethHeaderData.destMACAddress, ETH_ALEN);

            send_result = sendto(s, buf, len, 0,
            (struct sockaddr*)&socket_address, sizeof(socket_address));

            if (send_result == -1) {
                fprintf(stderr, "Send failed");
            }

            printf("Package sent, timeAllowedToLive: %i\n",
            milliintervals[intervalselector] );
            fflush(stdout);
            ticks = 0;
        }
        usleep(500000);
    }
    return 0;
}
```

APPENDIX 4. Retransmission algorithm walkthrough step by step

(timeAllowedToLive)

loop cnt	ticks	intervalselector	ticksintervals[intervalselector]	ticks >= ticksintervals[intervalselector]	intervalselector < 4?	intervalselector++	millisintervals[intervalselector]	reset ticks?	sleep
0	0	0	0	0	yes	yes	1		
1	1	1	1	0	yes	yes	2		
2	1	2	2	4	no				
3	2	2	2	4	no				
4	3	2	2	4	no				
5	4	2	2	4	yes	yes	3		
6	1	3	3	20	no				
7	2	3	3	20	no				
8	3	3	3	20	no				
...	...	...	...	...	...	...	...	...	...
22	17	3	3	20	no				
23	18	3	3	20	no				
24	19	3	3	20	no				
25	20	3	3	20	yes	yes	4		
26	1	4	4	40	no				
27	2	4	4	40	no				
28	3	4	4	40	no				
...	...	...	...	...	...	...	...	...	...
62	37	4	4	40	no				
63	38	4	4	40	no				
64	39	4	4	40	no				
65	40	4	4	40	yes	no	4		
66	1	4	4	40	no				
67	2	4	4	40	no				
68	3	4	4	40	no				
...	...	...	...	...	...	...	...	...	...
102	37	4	4	40	no				
103	38	4	4	40	no				
104	39	4	4	40	no				
105	40	4	4	40	yes	no	4		

							1024	yes	500
							4098	yes	500
							4098		1000
							4098		1500
							4098		2000
							16384	yes	500
							16384		1000
							16384		1500
							16384		2000
							...	...	...
							16384		9000
							16384		9500
							16384		10000
							32768	yes	500
							32768		1000
							32768		1500
							32768		2000
							...	...	...
							32768		19000
							32768		19500
							32768		20000
							32768	yes	500
							32768		1000
							32768		1500
							32768		2000
							...	...	...
							32768		19000
							32768		19500
							32768		20000
							32768	yes	

APPENDIX 5. The Manufacturing Message Specification protocol

The Manufacturing Message Specification (MMS) is an object oriented Application layer messaging protocol and a comprehensive standard containing multiple objects and services. IEC has chosen to map the ACSI class models and service models to MMS because it

- has a proven implementation track record,
- provides the required information modelling methods and services,
- supports the complex naming of IEC 61850 objects, and
- can operate over the full TCP/IP or OSI stacks .

Despite its name, the MMS standard does not contain any manufacturing specific definitions. In fact, it is very generic. Mapping of the ACSI to MMS is quite straightforward. (IEC 61850-8-1 2011: 14; Schwarz 2008: 2, 4; Mackiewicz 2006.)

MMS contains object classes such as Domain, NamedVariable, NamedVariableList and Journal, and services such as read, write, report, download and read journal. The main MMS object is called *Virtual Manufacturing Device* (VMD), and it contains all the other objects. The VMD represents a virtual interface to a real device by hiding the underlying implementation details, as depicted in Figure 39. It represents the part of an application that a client can access—a client can browse all the objects of the VMD and read all the attributes through MMS services. The attributes of all objects together represent a self-description of the device. The self-description can thus be downloaded by a software tool. (Schwarz 2008: 4-7.)

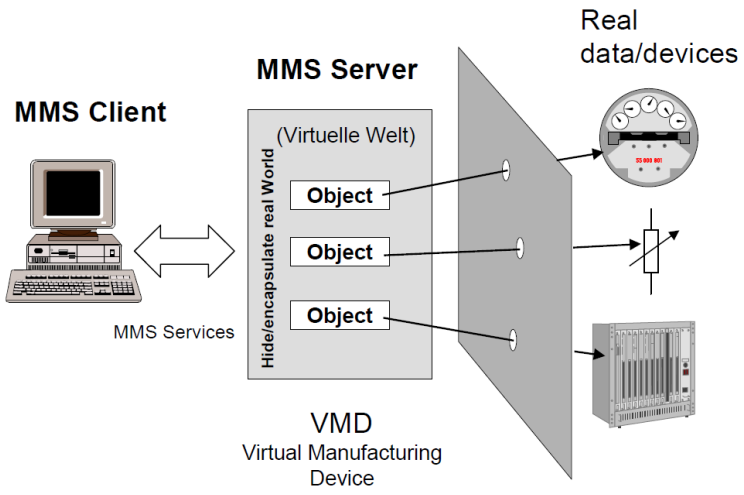


Figure 39. The VMD hides the concrete implementation of a device, i.e., it represents a virtual interface to a real device. A client can download the self-description of a device through MMS services. (Schwarz 2008: 6.)

The MMS environment is created through the establishment of a single TPAA, which is created and maintained via the client/server communication profile. An instance of an ACSI server class is mapped one-to-one to at least one VMD. Each VMD is assigned one or more communication addresses that each create a SAP through which MMS services can be exchanged. The mapping makes the VMD represent the capabilities of the server on the network. From the point of view of the client, only the server with its objects and its behaviour is defined and visible. The MMS server also contains files and client associations. (IEC 61850-8-1 2011: 32, 55, 58; Schwarz 2008: 5-6.)

An instance of a logical device is mapped to a MMS Domain object, which acquires the name of the logical device. An instance of a logical node, data object, data attribute, or control block class maps to a NamedVariable object, and a DataSet is mapped to a NamedVariableList, as given in Table 12. The NamedVariable class of MMS allows structuring any information provided for access by an application. (IEC 61850-8-1 2011: 32, 56, 65; IEC 61850-7-1 2011: 127; Schwarz 2008: 5.)

Table 12. Mapping of the objects defined in parts 7-2, 7-3 and 7-4 of IEC 61850 to MMS protocol objects. (IEC 61850-7-1 2011: 127.)

What to be mapped?	Maps to
Logical device (contains logical nodes); IEC 61850-7-2	MMS domain
Logical nodes (contains data); IEC 61850-7-4	MMS named variable
Data; IEC 61850-7-4	MMS named variable (and structured components of the named variable representing the "logical node data")
Data attribute; IEC 61850-7-3	MMS named variable (and structured components of the named variable representing the "data")
Data set: IEC 61850-7-2	MMS named variable list
Control blocks (attributes); IEC 61850-7-2	MMS named variable
Control blocks (behaviour); IEC 61850-7-2	Needs to be programmed as defined in IEC 61850-7-2
Log; IEC 61850-7-2	MMS journal

The content (semantic of the exchanged information) of the NamedVariables is not specified by the MMS standard, allowing the semantics of the ACSI class models to be used. The references of the MMS NamedVariables are created through the concatenation of the NamedVariable component names. Each level of hierarchy of the data model is separated by the $ character. MMS maps the FC value to the object reference, between the logical node name and data object name. As an example, the reference of an instance of the data attribute stVal of a circuit breaker is created through the concatenation of several MMS NamedVariable objects residing at different hierarchy levels, as illustrated in Figure 40. (Schwarz 2008: 5; IEC 61850-8-1 2011: 34, 93.)

Although MMS has many technical advantages, it fails to meet the requirements set for implementing GOOSE, mainly due to its lack of support for the publisher/subscriber architecture and its relatively slow performance. (Ozansoy 2006: 34, 122.)

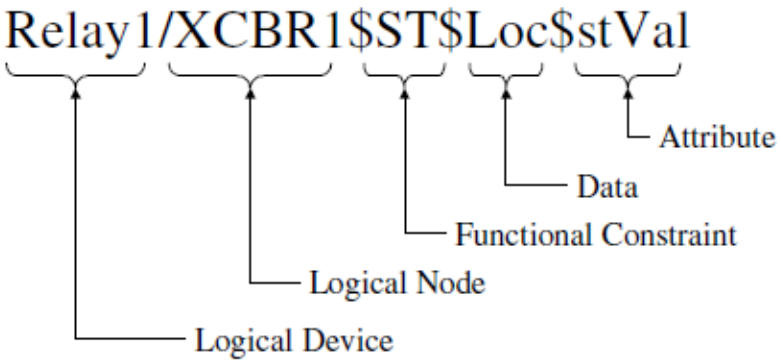


Figure 40. Example of an object reference in the MMS domain, created through the concatenation of the NamedVariable object names at different hierarchy levels. Unlike the object references in the IEC 61850 domain, the FC is mapped between the logical node and data object names, and object names are separated by the $ character. (Mackiewicz 2006.)

APPENDIX 6. Settings and bugs in Rapid61850

- Timestamping was turned on by setting TIMESTAMP_SUPPORTED to 1 in ctypes.h.
- The type of variable len in function gseEncodePacket in gseEncodePacket.c had to be changed from int to unsigned short. This is probably due to the properties of the LDU.
- Wrong endianness in ctypes.c caused the Application ID of GOOSE messages to acquire erroneous values and was therefore fixed by commenting:

```
// copies bytes to network format (big-endian)
void netmemcpy(void *dst, const void *src, unsigned int len) {
//#ifdef LITTLE_ENDIAN
//  reversememcpy((unsigned char *) dst, (const unsigned char *) src, len);
//#else
    memcpy((unsigned char *) dst, (const unsigned char *) src, len);
//#endif
}

// copies bytes to host format (little-endian)
void hostmemcpy(void *dst, const void *src, unsigned int len) {
//#ifdef LITTLE_ENDIAN
//  memcpy((unsigned char *) dst, (const unsigned char *) src, len); //
//#else
    reversememcpy((unsigned char *) dst, (const unsigned char *) src, len);
//#endif
}
```

  LITTLE_ENDIAN could not be defined because a source file linked in by the compiler already defined LITTLE_ENDIAN. It is therefore an LDU and Denx cross-compiler related problem.

- It was discovered that Rapid61850 sets the default VLAN identifier to 4 in gse.c. The problem was solved by setting the value to 0.
- Function gseDecodeDataset in gseDecode.c: The gocbRef for VAMP was incorrectly mapped by Rapid61850 and used the name of the logical device in IED Laptop: "VAMPC1/LLN0GOgcb1" was corrected to "VAMPRelay/LLN0GOgcb1". Also the value of gocbRefLength (19) had to be changed to 22 to match the amount of characters in the corrected gocbRef.

- In the SCD code, IED VAMP has two data attributes in its DataSet (virtual inputs corresponding to buttons F1 and F2 on the front panel of Vamp 50) the values of which are sent as a GOOSE message when either button is pressed. IED Laptop has an input that references only one of these data attributes (F1). However, IED Laptop can read the values of both data attributes in the received GOOSE message.