UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Anssi Jäntti

VISUALIZATION OF PROTECTION RELAY FUNCTIONS

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, February 23, 2012.

Supervisor	Jouni Lampinen
Instructor	Olavi Vähämäki

FOREWORD

"The Answer to the Great Question, of Life, the Universe and Everything
Forty-two
I checked it very thoroughly, and that quite definitely is the answer.
I think the problem, to be quite honest with you, is that
you've never actually known what the question is."
–Deep Thought, The Hitchhiker's Guide to the Galaxy
by Douglas Adams

I would like to express my gratitude to all of my friends, family and coworkers who have in any way helped me with my studies and made it possible for me to complete this thesis. I would like to thank Vamp Ltd. and especially my instructor Olavi Vähämäki for providing me with work during my studies and ultimately for providing me with the subject for this thesis and the opportunity to work on it full-time. I would also like to thank my thesis supervisor Jouni Lampinen for continuously providing me with meaningful feedback which helped and encouraged me to improve my work.

Last, but most definitely not least, I would like to express my deepest and sincerest gratitude to my parents, especially to my mother. Without their continuous and completely selfless support this work simply would not have been possible.

Anssi Jäntti
Vaasa, Finland, 23th of February, 2012

TABLE OF CONTENTS **page**

SYMBOLS AND ABBREVIATIONS

Abbreviations

API	Application Programming Interface
C#	C sharp, programming language
CPU	Central Processing Unit
FTDI	Future Technology Devices International
GDI	Graphics Device Interface
GDI+	Graphics Device Interface, newer version
GPU	Graphics Processing Unit
HMI	Human-machine interface
HTML	HyperText Markup Language
IDMTL	Inverse Definite Minimum Time Lag
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

LIST OF FIGURES

LIST OF TABLES **page**

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Anssi Jäntti
Diplomityön nimi: Visualization of protection relay functions
Valvojan nimi: Professori Jouni Lampinen
Ohjaajan nimi: DI Olavi Vähämäki
Tutkinto: Diplomi-insinööri
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2005
Diplomityön valmistumisvuosi: 2012 **Sivumäärä:** 99

TIIVISTELMÄ:

Suojareleet ovat sähköntuotannon ja siirtoverkon komponentteja, jotka auttavat verkkoa käsittelemään vikatilanteita. Modernit suojareleet sisältävät kymmeniä suojausfunktioita, joista jokainen puolestaan edelleen kymmeniä asetteluparametreja. Oikeiden parametrien valinta on järjestelmän toiminnan kannalta erittäin tärkeää, mutta toisaalta hankalaa, sillä asetteluparametrien väliset riippuvuudet saattavat olla epälineaarisia ja vaikeasti hahmotettavissa. Kuvaajan olemassaolo helpottaisi usein asetteluarvojen suunnittelua merkittävästi.

Tämän työn tarkoituksena oli konstruktiivisesti osoittaa, että on mahdollista kehittää suojareleen asetteluohjelmaan suojausfunktioita visualisoiva toiminnallisuus. Kehitettyjen toiminnallisuuksien tarkoituksena oli parantaa suojareleen asetteluohjelman käyttöliittymää ja täten helpottaa suorajareleiden asettelua. Työn tavoitteiden saavuttamiseksi suunniteltiin ja toteutettiin rajapinta lisäosien liittämiseksi suojareleen asetteluohjelmaan. Rajapintaa hyväksi käyttäen toteutettiin myös distanssisuojauksen kuvaaja. Ohjelmiston kehittämiseen käytettiin sovellettua versiota vesiputousmallin mukaisesta ohjelmistonkehitysprosessista.

Työn tuloksena saatiin aikaan ohjelma, joka toteuttaa edellä mainitut toiminnot. Ohjelman soveltuvuus käyttötarkoitukseen varmistettiin iteratiivisella prototyyppimenetelmällä, jossa ohjelman käyttöliittymästä tuotettiin prototyyppejä, jotka annettiin asiantuntijoiden kokeiltaviksi. Asiantuntijoiden antaman palautteen perusteella kehitettiin jälleen uusia prototyyppejä, kunnes ohjelma saavutti vaaditun tason. Ohjelman toimivuus vaadituissa olosuhteissa osoitettiin suorittamalla ohjelmalle ja sen osakomponenteille testejä simuloidussa käyttöympäristössä käyttäen normaaleja ohjelmistotestauksen menetelmiä.

Kehitetty ohjelmisto on ensisijaisesti tarkoitettu tuottamaan distanssisuojauksen kuvaajia, mutta työssä kehitettyä järjestelmää on helppo laajentaa tuottamaan myös muiden suojausfunktioiden kuvaajia. Työssä toteutettu ohjelmisto suunniteltiin toimimaan ensisijaisesti työn teettäneen yrityksen ohjelmiston kanssa, mutta työssä käytettyjä periaatteita ja menetelmiä voidaan soveltaa myös muiden valmistajien laitteisiin.

AVAINSANAT: suojarele, distanssisuojaus, asetteluohjelma, käyttöliittymä

UNIVERSITY OF VAASA
Faculty of Technology
Author: Anssi Jäntti
Topic of the Thesis: Visualization of protection relay functions
Supervisor: Professor Jouni Lampinen
Instructor: Ms.Sc. (Tech.) Olavi Vähämäki
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Information Technology
Major of Subject: Software Engineering
Year of Entering the University: 2005
Year of Completing the Thesis: 2012 **Pages:** 99

ABSTRACT:

Protection relays are part of electricity generation and transfer networks, which help the network to deal with faults. Modern protection relays have tens of protection functions, which in turn each have tens of setting parameters. It is very important for the functionality of the system to select for the right parameters, but on the other hand it is also very difficult as the relations between the parameters can nonlinear and difficult to understand. A graph would often make it easier to design the setting parameters.

The objective of this work was to constructively show that it is possible to develop for a protection relay setting tool a new feature which visualizes the functionality of protection functions. The purpose of those features was to improve the user interface on the protection relay setting tool and thus ease the setting of the relays. For achieving the objective, an interface was developed, which allows for developing addons for the protection relay setting tool. The interface was the utilized for generating a graph for distance protection. An applied version of a waterfall software development processes was used.

The result of this work was a program which implements the previously mentioned functionalities. The usability of the program was ensured by using an iterative prototyping process, where prototypes of the user interface were produced and then given to experts for testing. The experts gave feedback about the program, the feedback was used to create new prototypes and the cycle was repeated until the program was satisfactory. Normal software testing methods were also used to design and conduct on the program and its various parts.

The developed software is primarily designed for creating graphs related to distance protection functions, but the developed system can be easily extended to cover the creation of other types of protection function graphs. The software developed in this work was primarily designed to work in conjunction with the software of the company which commissioned the work, but the principles and methods developed here could be applied to other manufacturers' devices.

KEYWORDS: protection relay, distance protection, setting tool, user interface

1. INTRODUCTION

The purpose of this work was to constructively show that it is possible to develop new features to a computer program called Vampset. The primary functionality of these new features was to visualize distance protection function and thus improve the usability of the program. Vampset (later in this thesis referred mostly as a relay setting tool) is a relay configuration tool developed by Vamp Ltd (later in this thesis referred as the target company).

For the purposes of achieving the task at hand, a computer program was developed. The developed program works in conjunction with the relay setting tool and produces online graphs of the distance protection function in a user interactive manner. A generic plugin-API was also developed and implemented for the relay setting tool. This API enables an easier implementation of even more new features and also makes it possible to implement the new features with many different kinds of techniques.

Due to the highly graphical nature of the program, the Microsoft .NET Framework and its graphical subsystem called Windows Presentation Foundation (WPF) were the tools chosen for implementing the drawing of the graphs. Used programming-language was C#, also developed by Microsoft.

Experiments and tests were performed in order to show that the developed program can in fact perform the required tasks; the program was able to operate with the relay setting tool and it was also able to produce graphical representations of the distance protection function. Usability of the program was ensured by using an iterative prototyping software development process, where prototypes of the program were created and then given to experts for evaluation. New prototypes were created based on the expert's feedback and the process was repeated until the program reached a level which was satisfactory to all.

1.1. Motivation

Vampset is a setting tool for Vamp protective relays which is very widely used internally by the company and by the customers. Vampset is however very old and as it was initially developed in the earlier days of the company when the products it was used with

were much simpler, because these facts it does not conform very well to the requirements presented by the newer high-end relays (Virtala & Holmlund 2009).

One of the disadvantages of the relay setting tool is the fact that as the modern relays have much more functionality that the older ones, displaying all of this information is a bit messy and the usability of the program is compromised. Another major disadvantage, according to the customer acknowledgments, is the "old-fashioned" look and feel of the tool. (Virtala *et.al.* 2009)

As the various protection functions have very many different parameters, deciding on the correct ones can be very hard. Relations between the set values and the resulting protection functionalities are not always very intuitive. The relay setting tool is implemented using very old techniques and this is a major limitation considering the visuality and appearance of the program. It was determined that in response to the customers' requirements, any visually demanding upgrades and perhaps an entirely new version of the relay setting tool should be implemented using newer tools which are up to today's demanding standards.

Updates to the relay setting tool were divided to several different smaller projects. The sub-task chosen as the subject of this thesis was the task to implement a new functionality to the relay setting tool, which would allow the program to display graphical representations of various protection relay functions. Primarily the distance protection function, but the work needed to be done in a such manner that in the future it could be easily extended to include various other graphs. Other possible updates and upgrades to the relay setting tool are described in the Vampset update plans by Vainionpää (2011) and Virtala *et.al.* (2009). These include for example a major revamping of the GUI and tabbed menu structures in the relay setting tool. Some of the other suggested upgrades have already been implemented and others might be implemented in other projects in the future.

1.2. Target company

The author of this thesis has been employed by the target company since the year 2006. In the year 2010 the author was given a task to start designing and implementing some

upgrades to the relay setting tool. It was decided that the author would implement this project as his master's thesis.

Vamp Ltd is a company which specializes on protection relays and arc-protection systems. The company was founded in 9th of November 1994 and has since then released several different protection-relays and various related products to the markets. Originally the company operated under the name Vaasa Electronics. (Vamp Ltd. 2009a.)

The brand-name VAMP comes from words Vaasa Arc Monitoring and Protection. In the beginning of year 2009 Vamp Ltd was bought by Areva T&D (Vamp Ltd. 2009b). During the year 2010 the ownership of the company was transferred to Schneider Electric (Schneider Electric Ltd. 2010). The head-office of the company is located in Vaasa and the company employs approximately 40 people (Vamp Ltd. 2009b).

Vamp Ltd operates worldwide and the company has made sales in over 50 different countries. Production of the devices has been completely outsourced so that the company itself could allocate all of its resources on R&D, sales and marketing. (Vamp Ltd. 2009a.)

200-series protection relays have for a long time been the company's flagship products. An example of 200-series protection relay, VAMP 255, is show on the right side of the

Figure 1. VAMP 255 and VAMP 50 protection relays

Figure 1. 50-series is the newest series of protection relays developed by the target-company. First 50-series protection relays were launched in the year 2008. Newest additions to the 50-series, VAMP 51, VAMP 52 and VAMP 55 were released in following year. An example of 50-series device, VAMP 50, is shown on the right side of the Figure 1. Different relays have different types of combinations of the available functionalities and they are also each intended for different applications. (Vamp Ltd. 2009a)

1.3. Overview of this thesis

The first chapter is a short introduction to the subject at hand and describes the target company in very general level. Theory behind this subject is covered in the second chapter. It contains details about protection relays, software architectures, software inside the protection relays and also some discussion about software engineering, software testing and the used tools. The third discusses related works done by the target-company and its competitors.

The fourth chapter describes the software development process which was used in this work. Fifth chapter describes in detail the system designed and implemented in this work, used techniques and algorithms are also described. All the performed experiments and software testing processes presented on the sixth chapter. Finally the conclusions and some ideas of improvement and suggestions for future work are presented on the last chapter.

2. PROTECTION RELAYS AND RELATED SOFTWARE

This chapter deals with the theory and different concepts related to the subject of this thesis. First this section gives a very brief and generic introduction to protection relays. Second section describes in more detail the distance protection function, as it is the one which is the most relevant from the perspective if this thesis. The theory and practices introduced in these sections apply to practically all of the protection relays in the markets, regardless of the manufacturer.

The sections in the middle of this chapter describes in more detail the specific system which has been developed by the target-company; the architecture of the software, how the various parts store information and communicate with each other. Many of the principles described might also apply to products from other manufacturers. The inner workings of the system in question are important, because the serve as the basis on top of which the system described later on this thesis were built on.

The information concerning the inner workings of the Vamp protection relays and the related software was acquired by examining the source codes, circuit diagrams and most importantly by monitoring the communication between the devices and programs by using a set of various low-level debugging tools.

After the description of the protection relay specific aspects, this chapter describes some theory related to software development and testing. Finally the chapter is concluded by describing some of the most relevant aspects of the programming tools used to achieve the tasks at hand; .NET, C# and WPF application programming interfaces (API).

2.1. Protection relay

Electric relay in general is some kind of device, which causes a predetermined change in one or more electrical circuits when some specific conditions are met in the observed electrical circuit (IEC 2002). Protection relay in turn is one component in the electricity generation and transmission systems, which purpose is to detect faults and to minimize the resulting damages caused to property and personnel. Protection relays also function to speed up the process of recovering from the faults. (Paavola 1964)

Continuity of the electrical power supply is very important, because interruptions in the complex processes using the electricity can cause major negative economical effects (Paavola 1964). However it is virtually impossible, or at least very uneconomic, to build a completely interference-free electrical grids, but it is more useful to focus on reducing and minimizing the damages cause by the potential disruptions (Mason 1956).

One of the potential failures in the grid is known as earth fault. Earth fault refers to a situation where some part of the grid which is normally separated from ground comes in contact with the ground (Paavola, 1964). One possible cause of earth fault is for example a tree which falls on the power line during a storm and causes the wires to break and fall to the ground. The lines can be protected from falling trees for example by building lines far away from forests or by building them significantly higher than the highest treetops. Usually it is however borderline impossible to build the electrical grids in a way that ground faults could not happen under any circumstances. Some unforeseen circumstances could for example cause the structures supporting the power-lines to fail even though there were no possibility that a falling tree would be able to cut the lines. If the lines were buried under ground, they can e.g. be affected by frost in the ground or they can be accidentally dug up by a excavator. Because of these facts, it is necessary to focus on trying to minimize both the likelihood of the faults happening and the damages caused by the faults when they eventually do occur. The ability to detect and located faults in the electrical grid eases the process of minimizing the damages. (Mason 1956)

Protection relays are attached to the electrical network by using current- and voltage transformers, which are used to measure the respective value on selected points in the electrical network. Under healthy conditions, the measured values have determined limits inside which they may vary. Shifting from these predetermined values indicates a fault somewhere in the network and after certain time-delay the relay issues a trip, which activates a switch that disconnects the faulty segment from the network. The amount of the network which is left unpowered by clearing the fault, should naturally be as small as possible. (Hewitson, Brown and Balakrishnan 2005)

In the early days of power system protection the relays were electromechanical. The

old electromechanical IDMTL (inverse definite minimum time lag) relays for example basically comprised of a magned, spinning metallic disk, two pieces of iron and some copper-wiring. On the other hand the modern protection relays are small computers with their own CPU, memory and programming. (Hewitson *et.al.* 2005)

2.2. Distance protection

Distance relays measure the current (I) and voltage (U) from a power line and compare their ratio to determine the direction and distance of the faults in the network, in relation to the point where the measurements are taken from. Distance relays are set to operate when certain preset ratio has been reached. The ratio of current and voltage,

$$\frac{U}{I} = Z \tag{1}$$

is also known as impedance. Impedance in turn can be thought of being a vector

$$Z = R + jX \tag{2}$$

which consist of resistance (R) and reactance (X) and thus it is natural to present distance protection functions on an R/X-graph, where X is the ordinate, R is the abscissa and the measurement point is the origin. (Blackburn 1987)

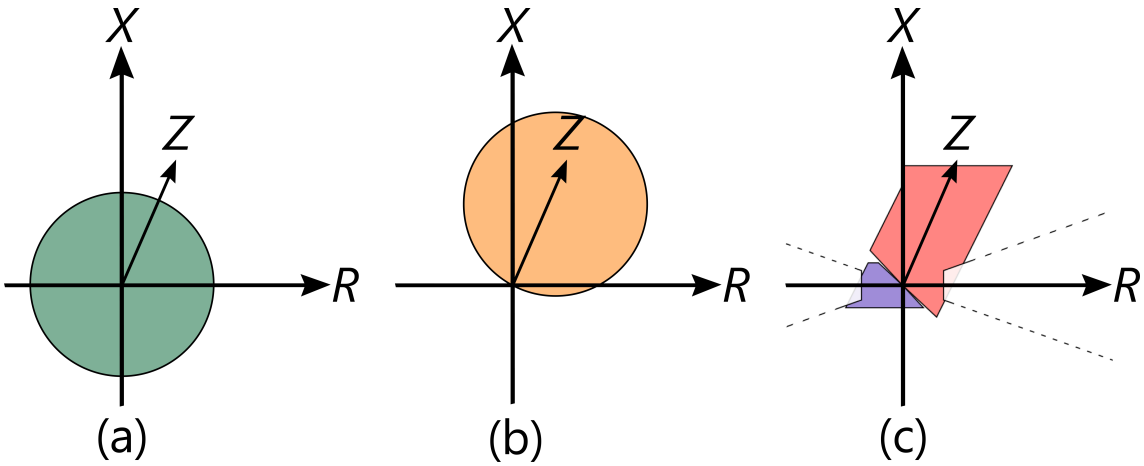

Figure 2. Examples of different distance protection graphs. (a) impedance, (b) MHO and (c) polygonal characteristic

There are numerous different ways for defining the impedance-values on which the distance relays should react. The most primitive method is just to define a circle which resides in the origin (shown on Figure 2a). This method originates from the first distance protection relays which were basically just two opposing coils (one controlled by the current-measurement and other by voltage-measurement) balancing a beam between contacts. This method only takes into account the distance of the fault, not the direction. (Blackburn 1987)

Another widely used method is called MHO characteristic (shown on Figure 2b), which extends the previously mentioned method by allowing the origin of the circle to be placed on different locations, in addition to defining just the size of the circle. This allows a configuration which takes into account the direction of the fault, in relation to the location of the protection relay. Additional parameters can also be available in order to prevent the relay from operating on normal load conditions. (Blackburn 1987)

The operating-area does not necessarily need to be defined as a circle; the available options depend on the relay manufacturer. Vamp protection relays determine a polygonal characteristics (example shown on Figure 2c). The polygonal characteristics on Vamp protection relays are defined by setting three parameters; resistance (R), reactance (X) and direction. These parameters define the size and the direction of the operating area. Resistance and reactance can be thought as parameters that affect how far the protection area reaches on the R- and X-axis. Fourth parameter, line angle, can be used to tilt the operating area. Two additional parameters, load angle and load resistance, can be used to define a blocking area which prevents the operation of the protection on high loads, which might in other cases be detected as faults. There are of course several additional parameters which affect the functionality of the distance protection function, but they are not apparent from this type of graph. One of them is the operation delay, which determines how fast the protection is activated when a fault has been detected. (Vamp Ltd. 2011)

The effects of which the various parameters on the Vamp protection relays have on the distance protection graph, are explained later in this thesis in more detail when discussing the rendering of the distance protection graphs.

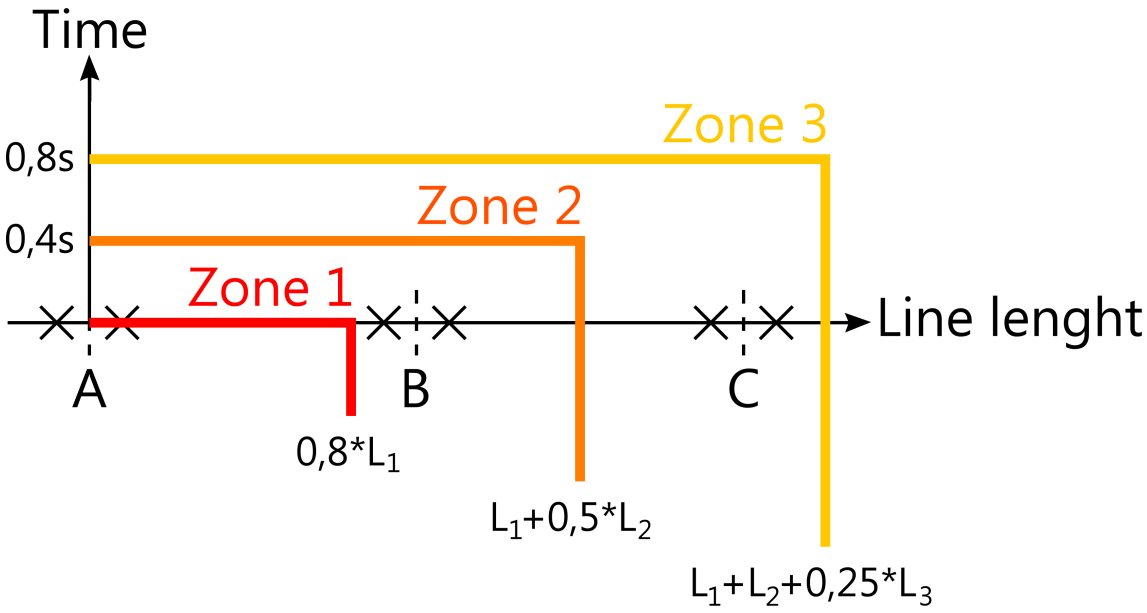

Figure 3. An example of typical settings for distance protection function and the zones.

There are also certain advantages of having several individual instances of distance protection functionality in a same location, because then the protection of the power transmission system can be considerably improved. The individual protection instances can reside on the same physical device, depending on the used devices. An example of distance protection zones is shown on Figure 3. X-axis represents the lenght of the power transmission line, where the power is being transmitted from left to right. Y-axis represents the time after detecting a fault in the power transmission line. Letters *A*, *B* and *C* below the X-axis represent the points in the line where the current and voltage measurements for the protection functionality are taken from. These can be e.g. electrical substations. Also the relays them self are usually physically located at the same places where the measurements are taken from. Location *A* resides on the origin and this is also the device (or group of devices) which is being configured. L_1 is the distance between locations *A* and *B*, L_2 is the distance between locations *B* and *C* and L_3 is the distance between locations *C* and the one following that. (Cook 1985)

On Figure 3 there are three distance protection zones configured to location *A*. The first is set to cover the line from location *A* to about 80% of the line to location *B*. Second zone

covers faults from location A to the middle-point between locations *B* and *C*. Finally the third zone covers faults from *A* to the line after the location *C*. Similar configuration exist at the other locations, i.e. the first protection zone at location *B* covers about 80% of the line L_2, *etc*. (Cook 1985)

The zone 1 (on Figure 3) is set to operate instantaneously when fault is detected. Full length of a power line is not covered in one zone, because of the possible errors in the measurements and settings. If for example the zone 1 would be set to cover the full length of the power transmission line between locations *A* and *B*, fault right after the location *B* might cause the protection at location *A* to react even though the fault should be cleared by the completely independent protection system at location *B*. (Cook 1985)

Unintended activations are prevented by not setting the first zone to cover the full length of the line (zone 1 shown on Figure 3). This would however leaves a small part of the line just before location *B* unprotected, but this is then taken care by setting second protection zone to cover the line from the location *A* to all the way to the middle point between locations *B* and *C*. The zone 2 is also set to operate after a time-delay so that it gives the protection at location *B* time to react for the faults occurring after the location *B*. Finally the zone 3 has been set as a backup for the line L_2 between locations *B* and *C* to serve as protection in case the line's primary protection at location *B* fails for some reason. (Cook 1985)

The example presented here is of course one of the simplest possible practical schemes. There is no communication between the different locations and any given location only looks for faults following itself. More complicated protection schemes can be implemented e.g. by setting all of the locations to look for faults on either direction and then communicate between different locations to determine and isolate the fault location fast and accurately. (Hewitson *et.al.* 2005)

2.3. Software achitecture

A commonly advocated way to layer computer programs is to divide them to three layers: user interface, application and database. Each part has its own individual purpose and

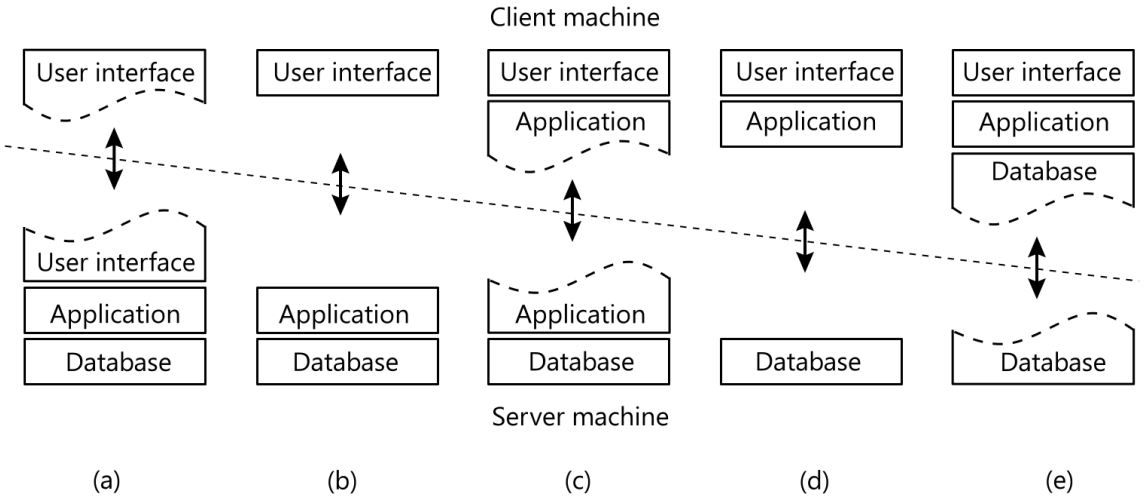

Figure 4. Common ways of distributing computer systems. Reproduction, original image copyright holder: Tanenbaum *et.al.*

the layers only interact with the ones right next to them; the database- and user interface-layers communicate with the application-layer, but the user interface and database do not communicate directly with each other. (Tanenbaum and Steen (2007)

In the commonly used three layered architecture, the user interface level contains the part of the program which interacts with the user. The user interface-layer e.g. reads the keys and displays menus and images to the user. The database level stores the data in the system and the application level sits in the middle, transferring and processing data. In the case of an distributed systems (where the system consists from separate client and server parts), the architecture of the program must be split to the parts which are located in the client and to those which are located on the server. There is a number of different ways to do the splitting, examples shown on Figure 4. For example the database can remain in the server machine, while the client has the user interface- and application levels. (Tanenbaum *et.al.* 2007)

Tanenbaum *et.al.* (2007) define distributed system as follows: "A distributed system is a collection of independent computers that appears to its user as a single coherent system.". A protection relay system could be thought as a distributed system in several different ways, depending on the perspective. From the perspective of the regular customer buy-

ing electricity, the whole electrical network appears as one huge distributed system. One transformer station could be considered to be an individual system in the electrical network, as stations typically have more than one protection relay.

The most interesting distributed system, in the concept of this thesis, is the combination of the protection relay and the relay setting tool. First of all, the software in the VAMP-relays follows the three-part layered architecture and it can be clearly divided into the three parts described previously. The relay software contains a database which is distinctly separated from the rest of the software. The database is used to store all the the information related to different kinds of protection functions, connection interfaces, menu structures, *etc*.

The most basic item, also known as a record, in the Vamp relay database is called DBitems. DBitems contain the stored value, type of the value (for example text, number or enumerated value), range of the value (for example from 1.0 to 10.0) and ID used to identify the items and separate them from each other. At the time of writing this, the combined database of all VAMP-relays has a total of 11856 unique DBitems. Of course different Vamp relay models only use a limited set of items which are required to the specific relay's operation.

VAMP-relays have a HMI-part (Human-Machine Interface) located in the front panel of the relays. HMI can be used to set and modify different DBitems using a convenient menu structure. HMI-part of the relay falls clearly to the user interface level of the common computer software architecture.

Last of the three parts of the described architecture is the application level. Application level of the relays contain the logic processing the measured values and activating the appropriate protection functions accordingly.

The combination of the relays and the relay setting tool can be considered as distributed system. The relay setting tool running on users computer can be considered as the client and the relay as the server. According to the current architecture implemented in the system, it is clear that in this case the whole database level is located in the server (the relay) and the user interface level is located in the client (computer running the relay setting tool).

The location of the application level in this system however might not as apparent. The relay setting tool can however be considered as a kind of extension of the HMI located on the front panel of the relay, or as a kind of remote HMI. While taking this approach, it becomes quite obvious that the system can be considered to be divided like the case (b) on the Figure 4 shows.

There is no actual technical documentation or description of this system so it is not clear how the initial designers of this particular system have intended this to be viewed. This is just the personal interpretation of the author of this thesis. There are probably a number of other possible approaches, among which is one where just the process where the relay setting is used to set the values would be considered as an independent system, separate

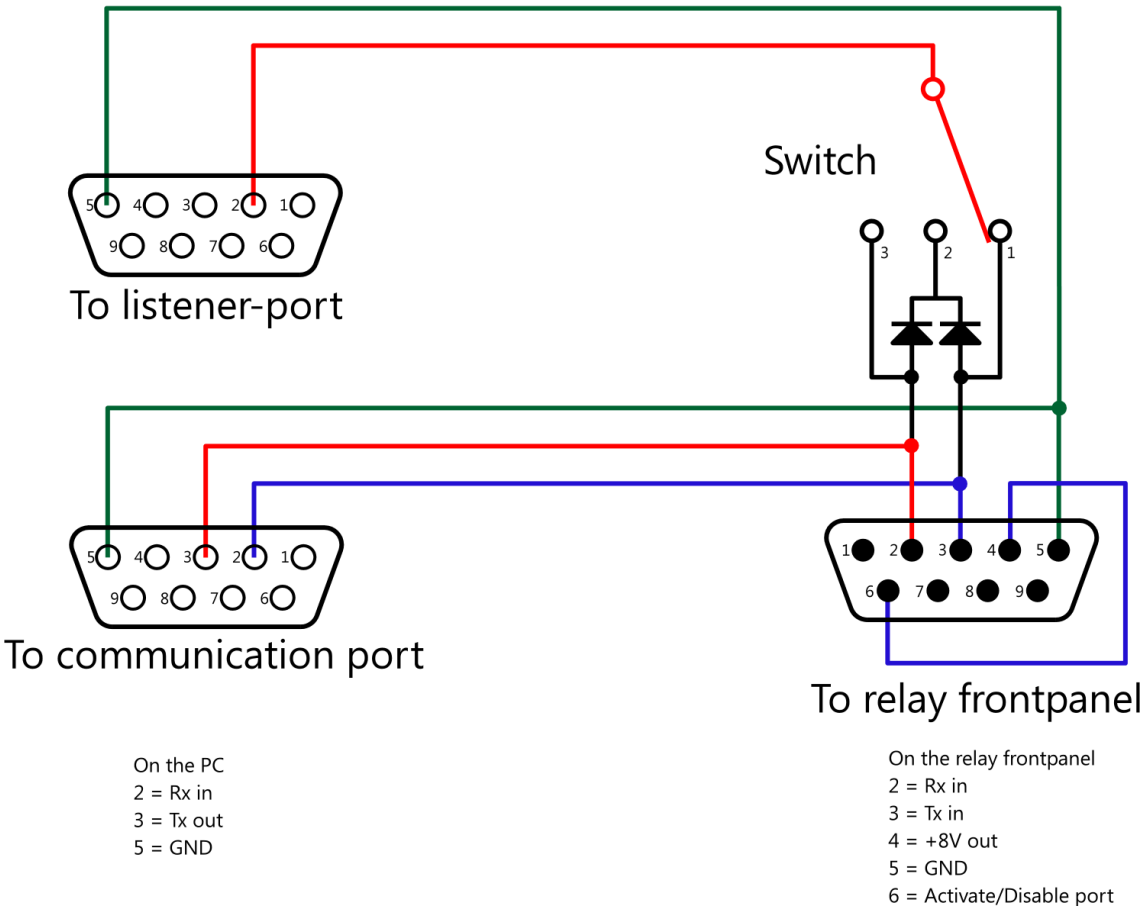

Figure 5. Diagram of the cable used to acquire information about the communication between Vampset and protection relays.

from the application where the relay is acting as a protection relay. This would result in a viewpoint where there are two independent applications using the same database, one application being distributed and the other located statically on a single device.

2.4. Serial communication and database

This section describes how the database inside the VAMP relays is structured, how Vampset uses it, how it can be accessed manually and how the VAMP-relays and the Vampset relay setting tool exchange data. The information presented here was mostly acquired by using a cable specifically made for listening serial communication between Vampset and Vamp relays. Construction of the said cable is shown on Figure 5. Results were confirmed from Kujanpää (software designer working at the target company).

During its time the communication protocol between Vampset and the relays has gone through some minor changes and because Vampset has to be able to communicate also with the older relays, it has been necessary to make several small fixes and makeshift solutions to the protocol on the side of Vampset. Because the system developed in this thesis should serve as a basis for the possible newer implementations of plugins and even the whole Vampset, these features are unnecessary there and thus they have been omitted from this description. A couple of bugs were also found while manually examining the communication between the relay setting and the relays. This purpose of this description however is not to list the found bugs but the correct behavior of the protocol and thus any bugs were also omitted here.

The listening cable (connection diagram shown on Figure 5) has three connectors: the 9-pin male D-connector is connected to the front panel of the relay, one of the 9-pin female D-connectors is connected on the PC to the COM-port which is used by the relay setting tool and the last connector is connected to the COM-port which is used to listen to the communication. The two female connectors are located on the left side of the Figure 5 and the one male connector is located on the lower right corner. The listening cable also has a switch which is used to select the signals which are being tapped. In the first position the cable listens to the messages sent by the PC to the relay. In the second position the

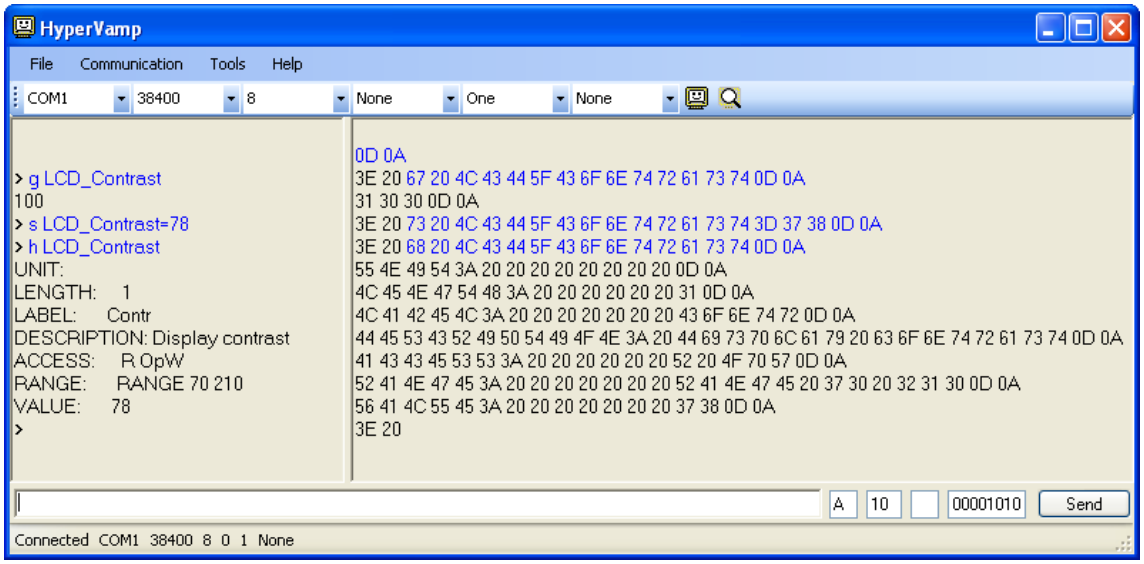

Figure 6. HyperVamp, a serial terminal for interfacing with VAMP protection relays.

cable listens for both the signals sent by PC and by relay and in the third position just the signals sent by the relay to the PC.

A terminal software developed by the author of this thesis was used for communicating with the relays and for logging the communication. The software is called HyperVamp and it is shown on Figure 6).

2.4.1. Serial communication

Serial communication is a way for computers and electronic to communicate and exchange data. In parallel data transmission the data is transmitted by using multiple wires simultaneously, while in serial communication the information is transmitted in succession along one wire. UART (Universal asynchronous receiver/transmitter) in turn is a device that acts as an interface between the parallel- and serial communications. RS-232 is a serial communication standard, which is defined in the EIA standard RS-232-C (Electronic Industries Association 1969). In desktop computers the interfaces implementing the RS-232 standard are commonly known as COM-ports.

USB is a bit more modern computer serial communication standard for communications between computers and various electronics devices. USB uses a master/slave-principle

where one party (the host or master) is usually the computer and the other party (the client or slave) is an accessory attached to the computer. The definition is not absolute, because e.g. some devices such as digital media players or printers can also act as the master.

FTDI Ltd. (Future Technology Devices International) manufactures integrated circuits, which in turn function as an interface between the older UART and the newer USB. One of FTDI's USB/UART-conversion slave-side circuits is called FT232R and one master-side product is called VNC1L. (FTDI Ltd. 2009)

2.4.2. Database

In addition to the local HMI on the relay's front panel and the PC-software Vampset, users can access the database using any common serial port terminal application (e.g. HyperTerminal). Communication between the relay and terminal software is done using Get/Set-protocol. Incidentally Vampset uses the same protocol when communicating with the relay. Vampset however does no display the exchanged messages to the user so this might not be apparent.

Using the Get/Set-protocol is fairly simple. To read a value of some parameter from the database, user simply writes the letter 'g', empty space and the ID of the DBitem in question. Setting values is done in similar manner; user first writes letter 's', empty space, ID of the item, equality sign and new value for the item. As an example, the following listing gives the command for reading the current contrast of the relay's front LCD-screen and the command for setting the contrast to a value 78:

```
g LCD_Contrast
s LCD_Contrast=78
```

Additional empty spaces can be placed between the different parts of the messages to make them more easy for humans to read, but they are not required. The letters 'g' and 's' can also be substituted with words "get" and "set". Relay also ignores capitals, so it

does not matter if any of the letters (the commands, IDs and values) are majuscules or minuscules.

In addition to the get- and set commands, there is third command called help. Help-command is used to display additional information about the DBitem in question. The listing below shows an example of the relays response, when the help-command used on DBitem `LCD_Contrast`:

```
> h LCD_Contrast
UNIT:
LENGTH:       1
LABEL:        Contr
DESCRIPTION: Display contrast
ACCESS:       R OpW
RANGE:        RANGE 70 210
VALUE:        78
```

In accordance with the get- and set-commands, the letter 'h' on the help-command can also be substituted with word "help", but in this case also with question mark '?'. All the rules regarding whitespaces and letter sizes also apply to the help-command.

While considering the subject of this thesis, the most important parts of the previously shown printout are the fields `DESCRIPTION`, `ACCESS`, `RANGLE` and `VALUE`. Firstly, the

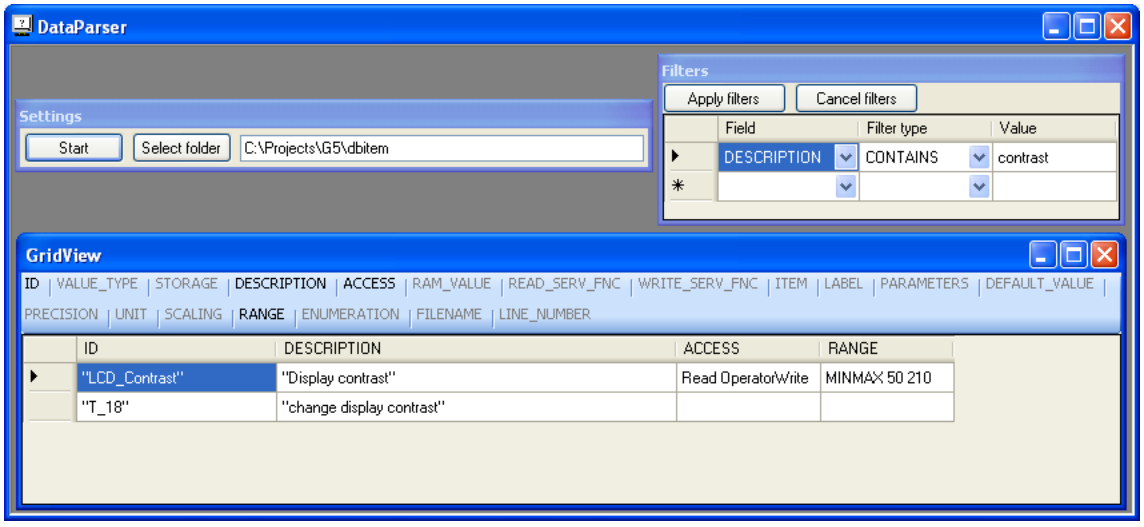

Figure 7. DataParser, an auxiliary tool for aiding the VAMP software development.

field DESCRIPTION provides a human-friendly description of the DBitem in question. The field ACCESS tells whether the type of the DBitem in question is read-write or read-only, and what access-level is required for using it. RANGLE-field lists the valid range of the following field; VALUE.

As described previously, all of the records available in the relay's database can be accessed if name of the DBitem in question is known. The names for the DBitems can be acquired by listening to the communication between the relay and Vampset, as described in the following section, but they can also be acquired by reading the relays software's source code.

A tool called DataParser (shown on Figure 7) was developed to aid the browsing of the available records. This tool takes as a parameter a file-path to the relay software's source code and it then parses all the records found in the database. Program then offers a way to search for items from the database.

2.4.3. Vampset communication

When the connection between the relay and Vampset is made, Vampset first sends a total of eleven commands to the relay. The commands, precisely as they are sent by the program, are listed here:

```
? AccessLevel
v
? Language
? ExtLan
? ApplOption
? Protocol
? ExtLedsOption
? VoltageMeasMode
? mAOptionParameter
? VampType
? ArcHW
? ArcType
```

First command the Vampset sends to the relay, is `? AccessLevel`. This queries the relay for information about the DBitem called `AccessLevel`. This item describes the current

access level, which in turn defines what DBitems the user is allowed to change. Access level can be changed by inputting a corresponding password.

The second command is just simply the letter `v`. This command instructs the relay to print information about the hardware available in the relay, relay's serial number and the version numbers of the installed software and hardware. Third command is `? Language` and it describes the language which the relay has been set to use. The fourth command `? ExtLan` gives the list of the installed and available languages. The fifth command `? ApplOption` describes the application-mode of the relay, e.g. motor protection or feeder protection. The sixth command `? Protocol` gives the protocol which is currently active on the Remote-port and also a list of available protocols.

The commands `? ExtLedsOption` and `? mAOptionParameter` give information about the possible additional LED- and analog output-modules if available. The command `? VoltageMeasMode` queries for information about the available voltage measurement mo-

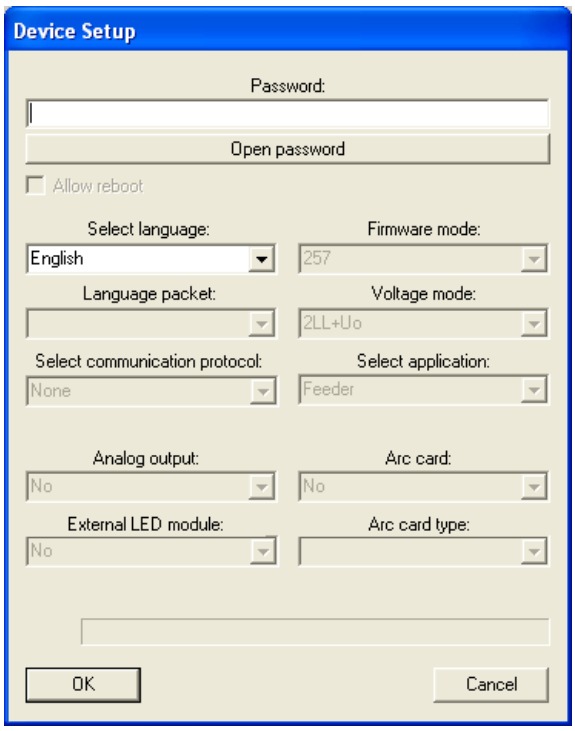

Figure 8. The "Device Setup"-window show by Vampset when connection is first established.

des and for which of them is currently selected. The tenth command `? VampType` gives the type of the device, e.g. Vamp 50, Vamp 255, Vamp 257, *etc*. The final two commands `? ArcHW` and `? ArcType` give information about the available arc-protection equipment.

All of this information is used to construct the "Device Setup"-window shown on Figure 8. The user of the program can use this window set the password and some of the parameters available in the target-relay. The user can click the "Open password"-button to apply the inputted password, which in turn enables those drop-down menus below, which corresponds to the access level of the given password. When the selections have been completed, the user can either click the "OK"-button to apply the changes or the "Cancel"-button to ignore the made changes.

When the "Device Setup"-window has been closed, Vampset sends to the relay a command `g MenuRoot[1]`. This produces a list of the menus available with the selected access level. Vampset then proceeds to read all the individual menus and creates the views for the user to read and modify the various parameters.

2.5. Software engineering

The term "software engineering" is described by the IEEE 610.12 standard glossary of software engineering terminology in the following way: "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.". (IEEE 1990) As this thesis is mainly about developing a relatively small piece of software, this is the part of the software engineering theory which receives the most focus.

This section first discusses about software development; the roles in the software development and about some software development process models. The second part of this section discusses about the software testing.

2.5.1. Software development

There are three distinct roles in the software development process: customer, user and developer. The customer is the organization or person who orders the work and is paying

for it. User is the party which will be using the end product and the developer is the one which does the actual implementation. Depending on the size of the software and various other circumstances, the different roles can be all be held by one party or they can be distributed amongst several organizations. The customer and user might for example be the same person. Various subcontractors and turnkey systems blend the borders between the roles, but they are always apparent at least on some level. (Shari Lawrence Pfleeger 2001)

A key factor in the software development is the communication between the various parties. The developer for example needs to know or find out what the user actually needs and what the customer is willing to pay for. Miscommunication might lead to a solution which is disappointing for all of the related parties. (Pfleeger 2001)

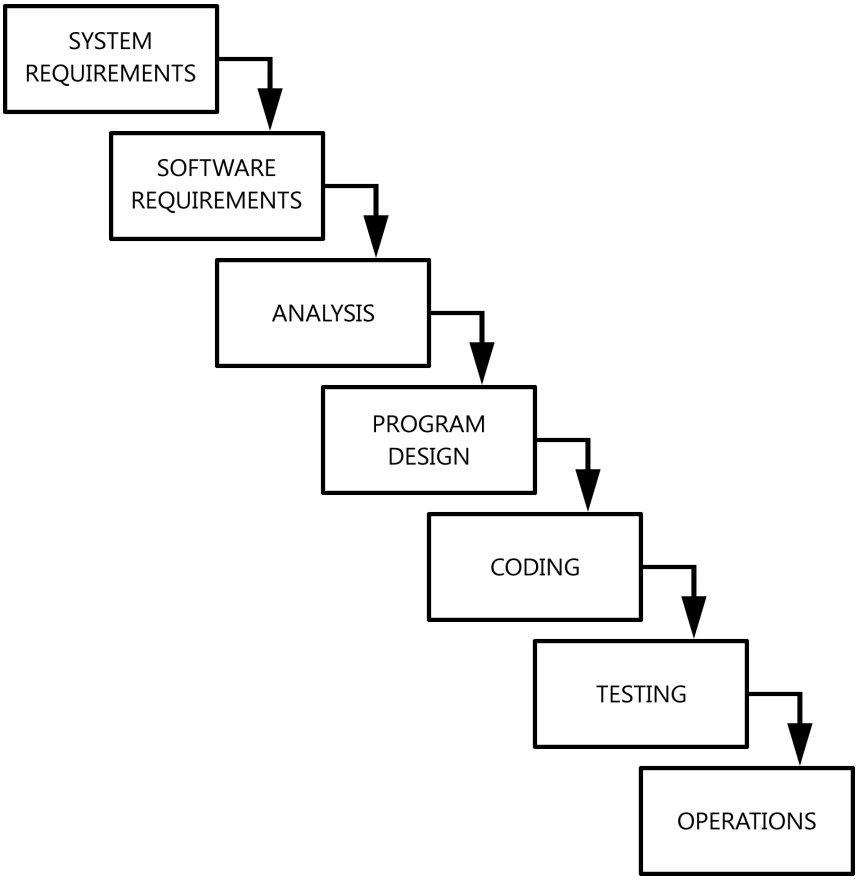

Figure 9. Waterfall model.

Perhaps the most commonly known and one of the first documented models for software development process is the waterfall model, which was first introduced by Winston W. Royce in 1970. The waterfall model describes the software development as a set of cascading steps, where one always leads to the following. (Pfleeger 2001) There are several different renderings of the waterfall model in the literature, but the waterfall model, as it was originally described by Royce (1970), is shown on the Figure 9. Royce actually never used the term "waterfall model" in the original article and the model itself was intended as a flawed way of developing software, to which Royce suggested ways of improvement.

The waterfall model might work on some cases where the problem and the setup at hand are very well known and defined. This is however very rarely the case in the software development. Some fundamental errors might for example be discovered in the testing-phase, which would in turn require changes in the requirements specification. The requirements might also even be initially incomplete. (Pfleeger 2001)

In the literature there is a lot of discussion about the problems with the waterfall model and about the solutions to these problems. Some solutions suggest changes to the waterfall model and others suggest completely different kinds of approaches and some of these include the V model, prototyping model, spiral model and transformation model. (Pfleeger 2001)

In the prototyping model a prototype of the system is constructed before the final implementation. Prototype is usually somehow incomplete compared to the final system. The prototype might for example only offer the user interface and the underlying functionalities might be crude or completely missing. After the prototype has been implemented, it might be scrapped and used as a basis for defining the new system or the prototype might be improved in order to use it as a part of the complete system. The prototyping model has been found to be particularly useful in the development of user interfaces. One major drawback in the prototyping models is that presenting the customer with a prototype might give a false impression that the system is almost complete even if majority of the system has not yet even been defined. (Ilkka Haikala & Jukka Märijärvi 2004)

The software development process models are not intended as absolute orders which must

be followed and to which the actual software development processes must be fitted to. More like to other way around: they are intended as guidelines and examples. The models can be tailored according to the needs of the individual software projects, developers, customers and users. (Pfleeger 2001)

No matter which method or mode is used, the following four elements are always present at least in some form or another: requirements specification, program design, coding and testing. Requirements are always acquired from somewhere. Program can not be designed if no requirements are available. Inherent aspects of software development are also obviously the designing and implementation. Especially in very small software development projects the coding might be started immediately after the requirements have been given, without much of the designing (Royce (1970). The designing and coding might be performed simultaneously, but usually it is suggested that at least some designing would be done before the actual implementation. The designing might be split to system-level design, program design and module design. Finally some testing is required in order to verify that the system performs as expected and fulfils the requirements. Testing might also include several different types and levels of testing. (Pfleeger 2001)

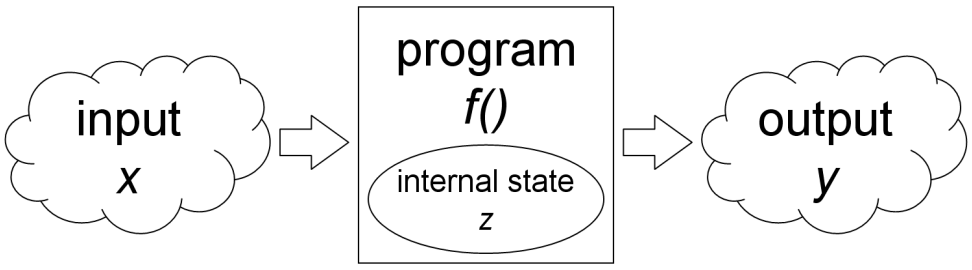

Figure 10. Computer program from the perspective of software testing.

2.5.2. Testing

The concept known as software testing is the process of running a computer program with an intent of finding errors. Manually browsing and reading the source code can also be considered as testing even though the program is not necessarily even being executed

at all. In addition to the inspections to the program or source code, software testing also encompasses the designing of the tests, building the testing environments and tools, reporting the findings and analysing them. (Haikala *et.al.* 2004)

Figure 10 illustrates how the computer programs (or just individual functions) are viewed from the perspective of software testing. The program $f()$ takes parameters as inputs, here denoted with x, which can for example be key presses, mouse clicks or data read from hard drive. Finally the program produces outputs y based on those inputs. In order to test whether the program functions properly with certain input, the required output has to be known in advance. Because of these principles, some kind of specification of the software is required before testing can commence. (Haikala *et.al.* 2004)

Proper testing can take a major portion of the resources reserved for the software development process, but when designed and done properly, it can be very effective. The amount of different possible input combinations is usually practically infinite even with small programs as programs tend to be indeterministic due to the human factors; the paths the execution and the internal states of the program varies greatly between different runs because of the users (Figure 10). Totally random testing is discouraged, and the tests should be planned in advance while taking into account possible problems and shortcomings. (Haikala *et.al.* 2004)

Verifying that a program has no errors would require the tester to test the program with all possible input combinations and as it is usually impossible this can't be viewed as the target of the software testing. The tester can never be sure that all errors have been found and even if the last error is found there is no way to know that it was the last. The main purpose of software testing is to improve the quality of the software by finding and finally repairing errors on the program before the program is released and the errors are found by the end-users. (Kaner, Falk & Nquyen 1999)

Even though planned testing is suggested, Kaner, Bach and Pettichord (2001) encourage testers for using exploratory testing and trust their instincts. As discussed in the previous chapters, the flow of the software development does not usually follow a straightforward model such as waterfall model, but is more chaotic as the requirements are changed and

new features are being added on the fly. If the software and specifications are constantly changing it can be hard and toilsome to keep constantly planning new formal test cases. (Kaner *et.al.* 2001)

Two opposing circumstances under which the software testing can occur are the black box testing and white box testing (also known as glass box testing). In black box testing the tester is not aware of the inner workings of the program. Black box testing is what most testers are doing for the majority of the time. In the white box testing the source code of the program is available for the tester. It is considered a common good practice for the programmers to routinely run white box testing for their own code as a normal part of programming process. (Kaner *et.al.* 1999)

There are several different types of measurements which can be used for evaluating whether or not the program has been tested enough. These measurements can also be used as basis for designing the tests. It should however be again noted that any level of testing does not fully ensure that there are no errors left in the program. Somewhat different methods can be used with white box and black box testing. (Kaner *et.al.* 1999)

One well known white box testing measurement is the test coverage. There are several levels of coverage and the weakest one is known as the line coverage, which measures how many lines of the code have been tested. Obviously ideally all lines of code should be tested at least once, however Kaner *et.al.* (1999) state that on average programmers do not employ even this level of testing. Two stronger levels of testing are known as the branch coverage and conditional coverage. Kaner *et.al.* (1999) also point out that all errors can't be found even if every possible line of the code and each decision-making function is tested with all possible conditions. If for example one of the input parameters is used as divisor; the calculation might be set as a non-zero value during the testing and the line would thus be marked as tested but ultimately it might fail with value 0.

Another way of defining the tests is known as equivalent classes. The idea here is that the input variables are divided to groups which are supposedly handled similarly by the program and thus only few candidates from those classes need to be tested. If for example the requested input is a number between 1 and 99 then one equivalent class are the

numbers between 1 and 99. Other classes could for example be numbers above 99, numbers below 1 and non-numerical characters. Another assumption is that boundaries of the equivalent classes are usually the most error-prone places and if the program fails at inside the borders then it usually also fails at the borders. Thus it makes sense to focus the testing on the borders of the equivalent classes. The software testing methodology of arranging the input variables to equivalent classes can be used with both the white box and black box testing. (Kaner *et.al.* 1999)

As it is common practice to divide bigger programs into smaller, more manageable and somewhat independent modules, it is also natural that these modules would be tested independently. This is called module testing or unit testing and it usually requires implementing additional testing tools as the individual modules can't otherwise be executed without the rest of the program. Another phase of the testing process is known as integration testing, where the individual parts are combined and tested as a whole. If the programmers want to save themselves from module testing, it is also possible to skip the module testing completely and move directly to the integration testing by doing what is known as the Big Bang integration. One major drawback with the Big Bang integration is that the sources of the problems are much harder to pinpoint from the complete system than it would be by testing modules separately. (Kaner *et.al.* 1999)

2.6. .NET, C# & WPF

.NET Framework is a software framework developed by Microsoft. .NET Framework includes a large library which provides a wide variety of functionalities, which can in turn be used to develop applications for various PC-, mobile- and web-environments. There are several different programming languages which can be used to access the features provided by the framework in question, some of these languages include C#, C++, Visual Basic and JScript. (Watson, Nagel, Pedersen, Reid, Skinner & White 2008)

When an application has been written by using the .NET Framework and is compiled, it is first compiled to Microsoft Intermediate Language (MSIL). This is a platform independent language and the purpose of this is to avoid optimizing the executable to different

platform as the configuration of the environment and the CPU-architecture can be completely different on the development-machine and on the machine where the program is ultimately used. When the program is executed, Just-in-Time (JIT) compiler compiles the program to the native code which is specific for the environment where the program is being executed. (Watson *et.al.* 2008)

C# (sometimes written C Sharp) is a programming language specifically designed for the .NET Framework and thus allows the developers to access all of the even most advanced features provided by the framework, which might be out of reach when using the framework with other languages. C# is based on the C and C++ languages. The purpose behind designing of the C# language has been to take the best qualities from its predecessor (C/C++) and make it easier to approach, while removing problems of the older languages and retaining the power of C++. (Watson *et.al.* 2008)

Windows Presentation Foundation (WPF) is a presentation framework, Microsoft's new graphics API, successor to the older GDI and GDI+ (Graphics Device Interface). WPF incorporates the capabilities of its predecessor GDI and the widely used internet markup language HyperText Markup Language (HTML). WPF has also been heavily influenced by Macromedia Flash. (Sells & Griffiths 2005)

One of the most prominent features of WPF is the separation of the user interface and the functionality. The user interface is designed with Extensible Application Markup Language (XAML) while the functionality and logic can be designed with C#. This clear division simplifies the application development and provides more freedom in teams where different persons are responsible of the user interface design and application logic programming. WPF also enables the usage of graphics processing unit (GPU) through DirectX in desktop applications, speeding up the usage of advanced graphical capabilities, while compared to for example GDI+. (Watson *et.al.* 2008)

3. RELATED WORK

Most likely all the different relay-manufacturers have their own tools and some may even contain functionalities for visualizing the protection functions. Siemens for example has a relay setting tool called DIGSI 4 (Jachmann & Feuerer 2008), but because this tool is not freely available, more detailed analysis was not feasible in the scope of this thesis. However, according to the Siemens SIPROTEC System Description manual (2008) the tool can be used to visualize for example the distance protection functionality.

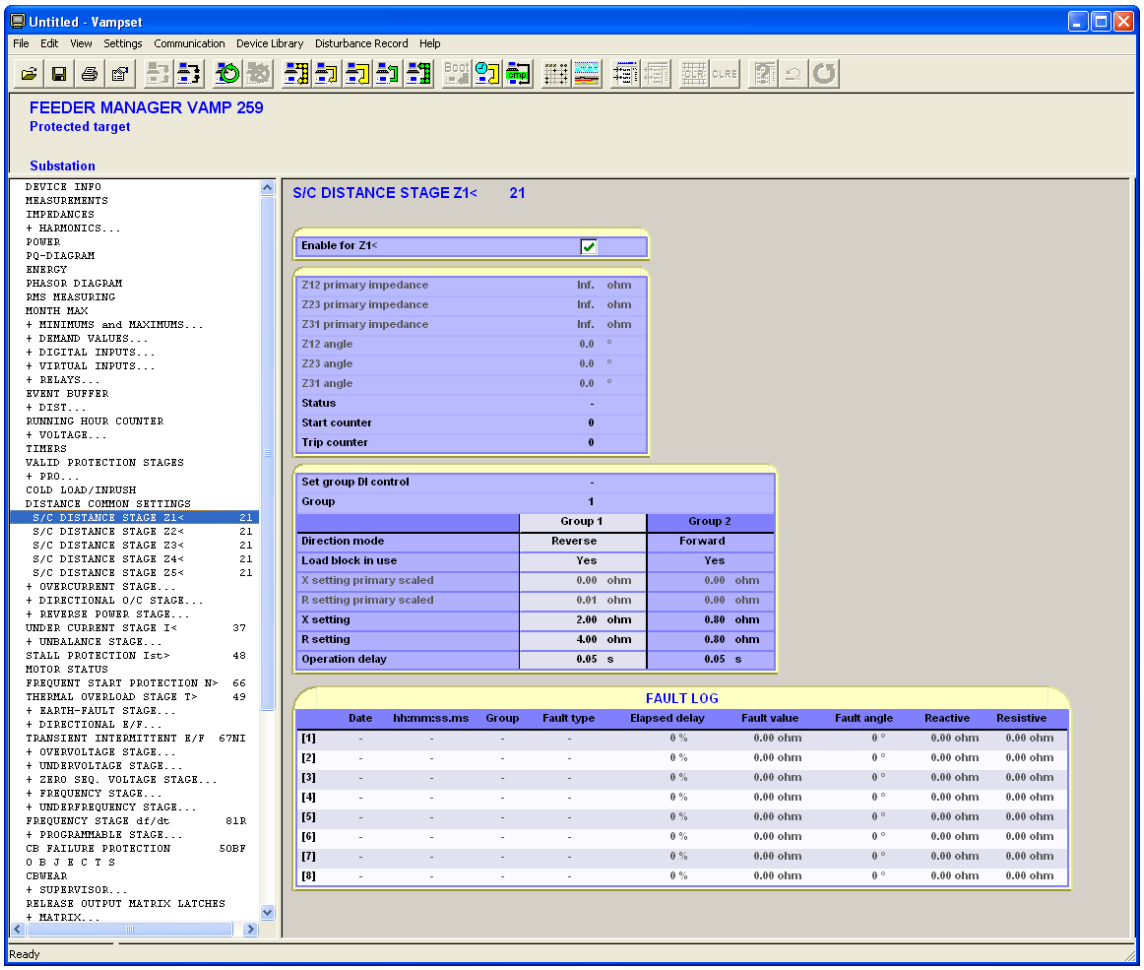

Figure 11. Vampset relay setting tool, configuring the distance protection function zone 1.

The tool used in the target company is called Vampset. The program is freely available for anyone to download from the company's website. The Vampset relay setting tool can however only be used with VAMP protection relays and even though it can be freely downloaded, it is not very useful for anyone who does not own a VAMP relay. The functionalities of the program can however be studied by using the accompanied example-files.

Figure 11 shows the view on the Vampset, which is used to set parameters for the first distance protection zone. As seen from the image, there is currently no graph or other visual aid for setting the distance protection. Setting is done only by changing the numeric values. If graphs are required, they need to be produced manually e.g. by drawing them on a paper by hand or plotting them with some custom software. This problem is to be remedied by the work done in this thesis. The inner workings of Vampset were explained in a bit more detail in the previous chapter.

4. SOFTWARE DEVELOPMENT PROCESS

This chapter describes briefly as a whole the software development process used while developing the software described in this thesis. The work done in the individual parts, such as the exact test cases selected for the different phases of the module testing, are explained in more detail in the following chapters.

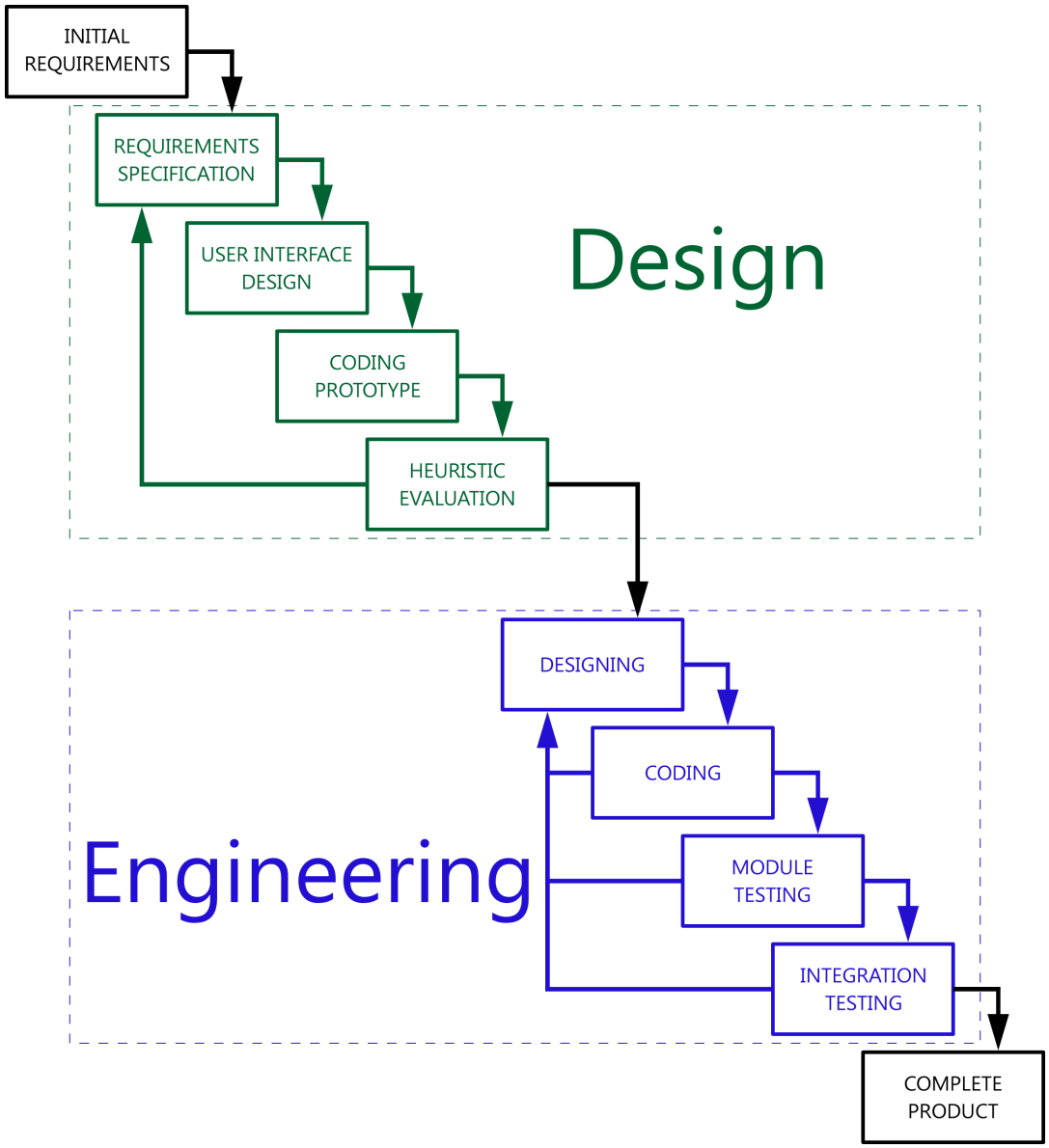

Figure 12. Description of the process used for developing the software described in this thesis.

The process required for developing this software did not directly correspond with any commonly known model and thus a new model was created. The model can be thought as a combination of waterfall- and prototyping models. The Figure 12 shows a diagram of this software development model.

The software development process used in this work was divided to two distinct parts, shown on the Figure 12. The first of these was a design process which was used for designing the user-interface part of the program. The second part was an engineering process during which the application logic itself was designed, implemented and tested.

The initial requirements were specified verbally by a person representing the target company, the first step on the Figure 12. A reference picture of the distance protection graph was also provided, the picture originated from the target company's user manual of the protection relay which had distance protection functionality. The initial requirements were first taken as a basis for the first draft of a requirements specification, second step on the Figure 12. As the requirements were informal and incomplete, they needed to be refined. This was done during the user interface design process.

After a first draft of the requirement specification was completed, a prototype of the user interface was designed, as shown on the third step on the Figure 12. Next phase was to code a functional computer program which implemented the user interface part of the program. This was the fourth step shown on the Figure 12.

After a prototype had been implemented, it was handed to an expert who was asked to try to use the prototype of the user interface as they would use the final product. The experts were asked to report any problems and requests for functionalities. The experts were also observed by the programmer during the testing in order to find out how the program would most like be used. This was shown as the fifth step on the Figure 12.

The feedback, comments and observations from the heuristic evaluation were then taken into account and incorporated back to the requirements specification, this part is shown as the path from the heuristic testing to the requirements specification on the Figure 12. After refining the requirements, a new functional prototype was programmed. The new prototype was then given back to the expert for verifying that the requests had been filled.

When the expert had accepted the new version of the prototype, the prototype was given to another expert for testing. This cycle was repeated five times; until all the requests were fulfilled, the services of all available experts were employed and everyone was satisfied with the program.

The first part of the whole process, the design process, was finished after the requirements specification and the user interface had been finalized. The second part of the process was the engineering process. This transition is also shown on the Figure 12.

The engineering part of the development process followed a bit more traditional waterfall software development model. The application logic was first designed, the programmed and tested. Different types of tests were performed and these included module- and integration testing. Black-box testing, white-box testing, equivalence partitioning and boundary value analysis techniques were used during the testing process. Module testing was mostly performed as a non-formal exploratory white-box testing and the integration testing was a series of more formal and planned test cases. The steps of the application logic development phase are shown on the lower part of the Figure 12.

When errors were encountered either during the coding or testing, the choices made in the application logic designing and coding phases were re-evaluated and this is shown as the backtracking arrows on the lower part of the Figure 12. Finally, after the program had passed the testing, a complete product emerged.

5. DESIGN AND IMPLEMENTATION

The two individual pieces of software developed and implemented in this thesis are the Vepset3 plugin API and the distance protection graph drawer. The Vepset3 plugin API was initially developed so that functionalities of the relay setting tool could be appended with visualizations of various protection relay functions, but it was decided that the API would be made as flexible as possible so that it would be able to support any other kind of plugins if required in the future. Using the Vepset3 plugin API, a visualization for distance protection function was also implemented.

5.1. Vepset3 plugin API

This section specifies the API through which the relay setting tool communicates with various plugins. Plugins in this context are considered to be programs which are running in conjunction with the relay setting tool and somehow utilize the data processed by it. Starting and closing of the plugins are controlled by the relay setting tool.

The main purpose of the plugins is to extend the functionality of the relay setting tool by allowing the usage of independent components and various implementation techniques. This plugin API is supposed to serve as a generic interface to the relay setting tool. Plugins are to be designed in a such manner that, besides the functionalities described by this API, they require minimal additional support from the relay setting tool.

This API firstly describes how the plugins are to be stored in the computer's memory. API also describes a configuration-file which must be provided in addition the the plugin executable, how they should be structured and where they must be stored. Finally this API describes the runtime communication between the relay setting tool and the plugins.

5.1.1. Plugin location

The actual executable binaries for the plugins can reside in any given location. They can for example either be on the hard-drive of the computer, on a network-drive or on a USB-memory stick.

Recommended practice is to place the plugin executable and all other related files inside some plugin specific folder. The name of that folder should be the name of the plugin in question. This folder should in turn be placed to a folder called "Plugins", which is located in the same folder as the relay setting tool's executable. So if for example if the executable of a plugin is called Vepset4.exe, the name of the plugin is Vepset4 and the relay setting tool is located in folder C: \ VAMP, the recommended path to the executable would then be C: \ VAMP \ Plugins \ Vepset4 \ Vepset4.exe

The plugin-specific folder can also be freely used by the plugin to store any other required files and even additional subfolders. The name of the plugins' executable does not have to match with the name of the plugin itself (and thus the folder where the plugin should be stored), however it is a recommended practice.

5.1.2. Configuration file

Each plugin must have a corresponding configuration-file stored in either the root of the "Plugins"-folder or in a sub-folder located in the "Plugins"-folder, which in turn is located on the same folder as the relay setting tool executable file (aka. Vampset). The relay setting tool reads all of the XML-configuration files located in the "Plugins"-folder at startup and uses them to determine how the plugins should be handled. An example of a typical configuration-file is shown below. All of the mandatory fields are shown in this example:

```
<?xml version="1.0" encoding="utf-8"?>
<Settings>
  <Vepset3>
    <Location>C:\VAMP\Plugins\Vepset4\Vepset4.exe</Location>
    <Port>52991</Port>
    <AutoStart>True</AutoStart>
    <Icon>True</Icon>
    <Menu>
      <Vepset4>
        <Settings />
        <Reload />
        <Reset />
        <Help />
      </Vepset4>
```

```
    </Menu>
  </Vepset3>
  <Plugin>
    <Position>0, 0</Position>
    <Size>800, 600</Size>
    <Mode>Embedded</Mode>
  </Plugin>
</Settings>
```

As an XML-file is used for the configurations, the file-extension of the configuration files must be .XML. The name of the configuration files can be freely chosen, except if the file is stored inside a subfolder of the "Plugins"-folder, the name of the configuration-file must match the name of the subfolder in question. A recommended practice is to use the name of the plugin in question.

In addition to the mandatory fields presented here, the configuration-file can also contain an undefined amount of additional elements and values which can be used internally by the plugin. These freely chosen fields should be placed inside the <Plugin/> and <Plugin> tags, after the mandatory fields. The relay setting tool ignores these additional elements completely.

The first line on the configuration-file is a standard XML declaration. The "Settings"-element on the second line begins the listing of the actual settings. Values inside the "Vepset3"-element are mainly used by the relay setting tool and values inside the "Plugin"-element are configurations for the plugin.

"Location"-element defines the location for the plugins' executable binary. This URI is processed with standard Windows conventions: path is treted as absolute if it begins with drive-identifier and as relative if the value begins for example with the backslash-character (\, ASCII-value 0x92d) or with a name of a file or folder.

"Port"-element defines the port for the socket which is used to communicate between the relay setting tool and the plugin. This element can have a default-value which the relay setting tool tries to use, but if the desired port can not be opened, the relay setting tool opens a random port from the range 49152–65535. If the port is changed, the relay setting

tool writes the number of the new port to this element. Plugins should never assume that the preset default-port is used. Number of the port must always be read from this element on startup.

"AutoStart"-element tells the relay setting tool that the plugin should be started automatically when the relay setting tool is loaded. Valid values include "True" and "False". If automatic start is disabled, the relay setting tool starts the plugin when one of the corresponding menu-items is activated or when user activates the icon corresponding to the plugin.

"Icon"-element determines whether the relay setting tool should display an icon for the plugin in the icon-menubar. If icon is enabled, an icon-file must be available in the same location as the plugins' executable binary. Name of the file must be the same as the configuration-file, except that the file extension must be "ico" instead of "xml". Valid values for this element are "True" and "False".

"Menu"-element contains a menu-structure for the plugin. This menu-structure is shown on under the "Plugins"-menu in the relay setting tool. The structure of the menu is a tree and it can contain several nodes and child-elements. When a menu-element is activated in the relay setting tool, a message is sent to the corresponding plugin. This field is not mandatory and when omitted, no menu for the plugin is shown.

"Position"-element defines the position on the screen to which the plugin should be opened. "Size"-element contains the desired size for the plugin-window. "Mode"-element tells the plugin whether it should be loaded in its own window, embedded in the relay setting tool (without borders) or completely without GUI. Valid values are "Window", "Embedded" and "Hidden". The relay setting tool modifies these values if required, so plugins should always read these on startup and obey them.

5.1.3. Communication

The relay setting tool communicates with the plugins through the configuration-file and sockets. Configuration-file is used to load startup-settings and sockets are used for runtime-communication.

Table 1. Messages available in the communication protocol between the plugin and the relay setting tool.

ID	Name	Description
0	exit	Closes the plugin.
1	position	Tells the plugin window to move to the provided position on the screen.
2	size	Tells the plugin to set its window to the provided size.
3	change	The relay setting tool tells the plugin that the current view has changed.
4	view	Plugin asks the relay setting tool to give all the data related to the current view. The relay setting tool answers with one massive blob containing all the data in the current view.
5	get	Sender asks the receiver to provide value stored in some specific database-item.
6	set	Sender tells the receiver to set a new value for some specific database-item.
7	bitmap	The relay setting tool asks for the plugin to provide a bitmap of the current view. This is also used when the plugin sends the bitmap as a response.
8	mouse	The relay setting tool tells the plugin that there has been a mouse-click on the previously provided bitmap, on specified coordinates.
9	menu	The relay setting tool tells the plugin that one of the plugins menus has been activated.
10	ping	The relay setting tool sends random 8-bit unsigned integer to the plugin and plugin has to respond to that by appending one to the number.
11	ack	Acknowledgement; a message indicating that the data has been received and the required actions were performed. This is used to terminate a message-flow.
12	nack	Negative acknowledgement; a message indicating that the data has been received, but was not acted upon. Appropriate actions should be taken.
13	invalidate	The relay setting tool tells the plugin a view was refreshed from a relay and the related information is no longer valid.
14	crcerror	An erroneous checksum was detected, last message must be re-sent.
15	state	The relay setting tool actives or hides the plugin.

When the relay setting tool starts a plugin, it gives the location of the corresponding configuration-file as arguments. Plugins should read the appropriate elements and values from the configuration-file and act accordingly. Before the relay setting tool starts the plugin, it starts to listen for incoming TCP/IP-sockets on the port described earlier. When the plugin is loaded, it should call the relay setting tool on the specified port. When connection is established, the relay setting tool first sends the ping-message (described later in more detail).

Runtime-communication through sockets has the following possible commands (command

IDs, enumerations and explanations) are listed on Table 1.

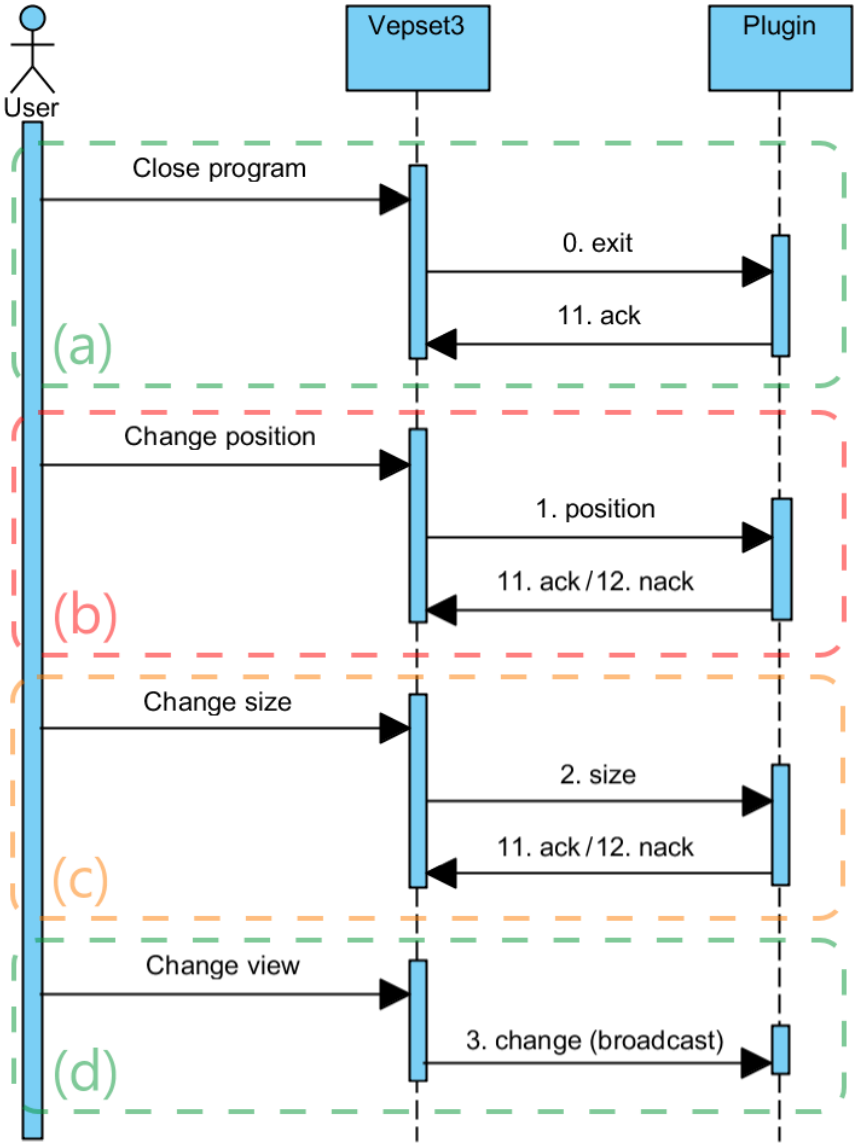

Figure 13. Message exchange diagram of the various messages exchanged between the relay setting tool and the plugins, part 1.

The relay setting tool sends the close-command to the plugin when the plugin needs to shut down. At this moment the plugin should also save its data to permanent storage, if required. Plugins should always obey this and shut down as soon as possible. Plugins should respond to this with an ack. If no ack is received, the relay setting tool closes down

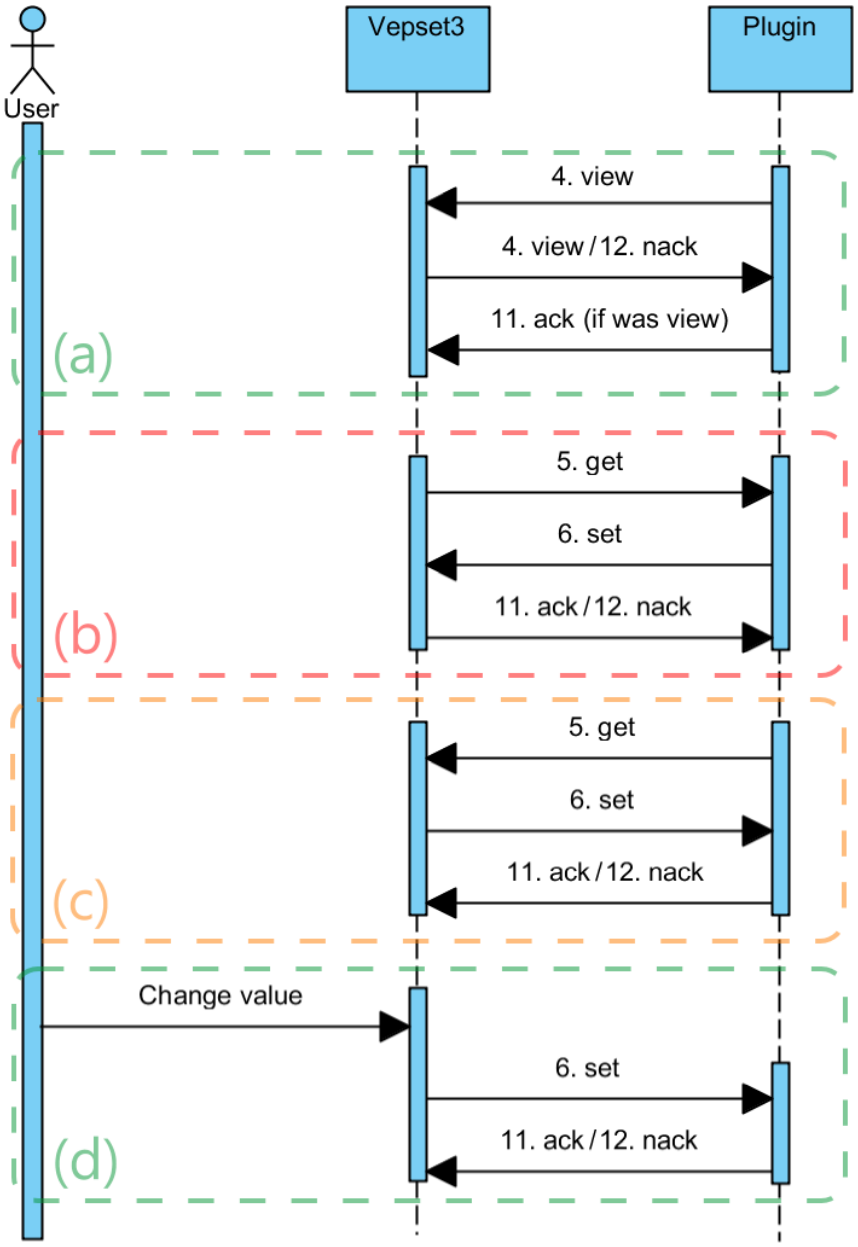

Figure 14. Message exchange diagram of the various messages exchanged between the relay setting tool and the plugins, part 2.

the plugin. Message exchange shown on Figure 13 case a.

Position- and size-commands are used by the the relay setting tool to move the plugin around the screen and change its size. This feature can be used to embed the plugins window to the relay setting tool window so that for the user they appear to be just one program. This feature requires special implementation on the side of the relay setting tool.

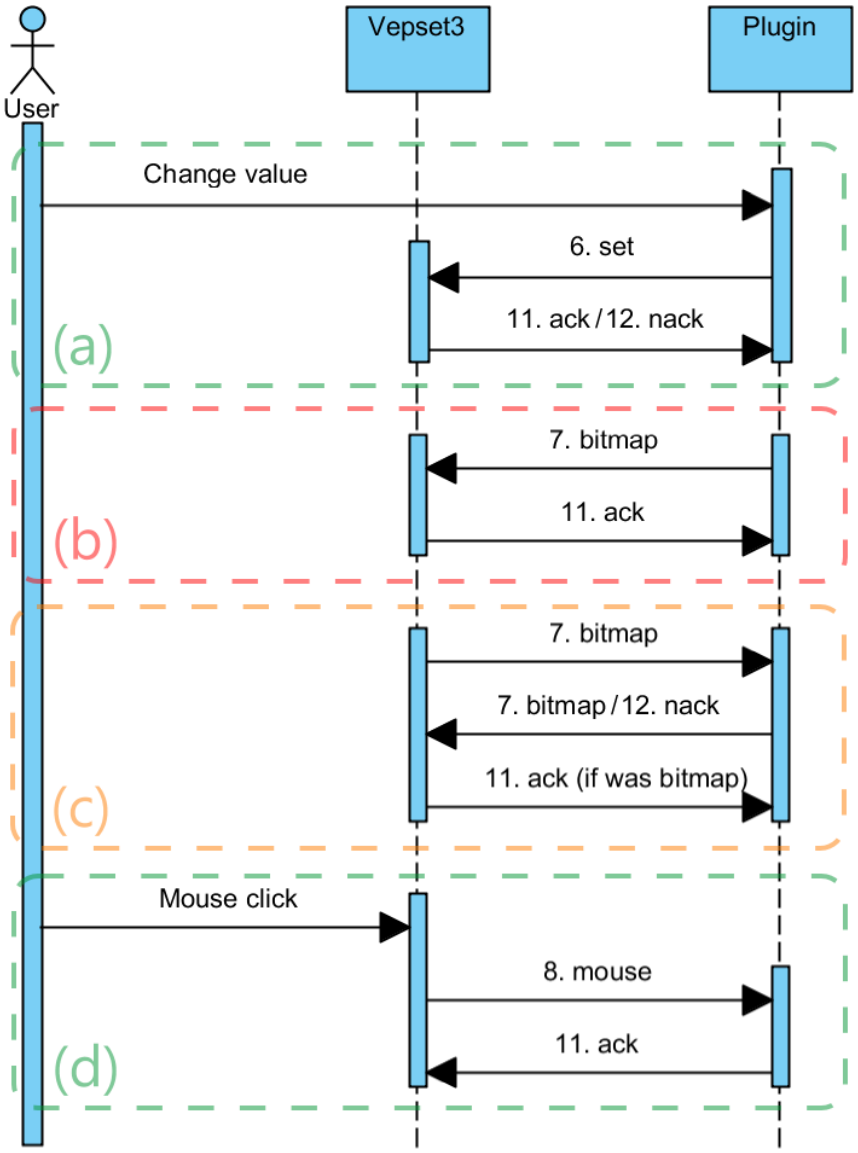

Figure 15. Message exchange diagram of the various messages exchanged between the relay setting tool and the plugins, part 3.

X- and Y-coordinates are represented with 16-bit unsigned integers, by using big-endian encoding. Origo for the coordinates is the upper-left corner of the screen. Plugin should respond to these commands with ack or nack. Message exchange shown on Figure 13 case b and case c.

When user changes the active view on the relay setting tool, the change-message is sent to plugins. The change-message is accompanied by the name of the activated view. This

message does not require an ack as a response, as this is a broadcast-type message and plugins can ignore this if they are not interested on the view in question. Message exchange shown on Figure 13 case d.

Plugins can use the view-command to request information about any specific view. If the view is available, the relay setting tool responds with one massive blob of data containing the required information about the view. If the relay setting tool was not able to produce the requested information, it responds with nack. Message exchange shown on Figure 14 case a.

Get- and set-commands are used to set a value to some parameter on the plugin and the relay setting tool. Message-field contains the data (the item name and possible new value) accompanied by the get- and set-commands. Both the plugin and the relay setting tool can set and get values from each other. An ack or nack must be sent as a response to the set-command, according to the result of the set-operation. A get-query should be responded with set if a response could be generated. If response to a get could not be generated, the response should be nack. Message exchange shown on Figure 14 cases b, c, d and on Figure 15 case a.

The relay setting tool uses the bitmap-command to query the plugin for a bitmap representing the view of the plugin. Size and the type of the requested bitmap can be provided as the message. X- and Y-dimensions are represented with 16-bit unsigned integers, by using big-endian encoding. Type of the bitmap is 8-bit long unsigned integer, following the sizes. Type must be set to 0, as other values are reserved for future use. If no size is specified, the previously defined size should be used. Plugin responds to this query with the requested bitmap on the message. Plugin can also respond with nack if it was unable to generate the bitmap. Plugin can also spontaneously send a new bitmap to the relay setting tool. The relay setting tool responds to these with ack. Message exchange shown on Figure 15 case b and case c.

Mouse-command is used to inform the plugin about mouse-clicks on the previously requested bitmap. Message should contain information about the location of the mouse-click and the button of the mouse which was clicked. Mouse coordinates should be rel-

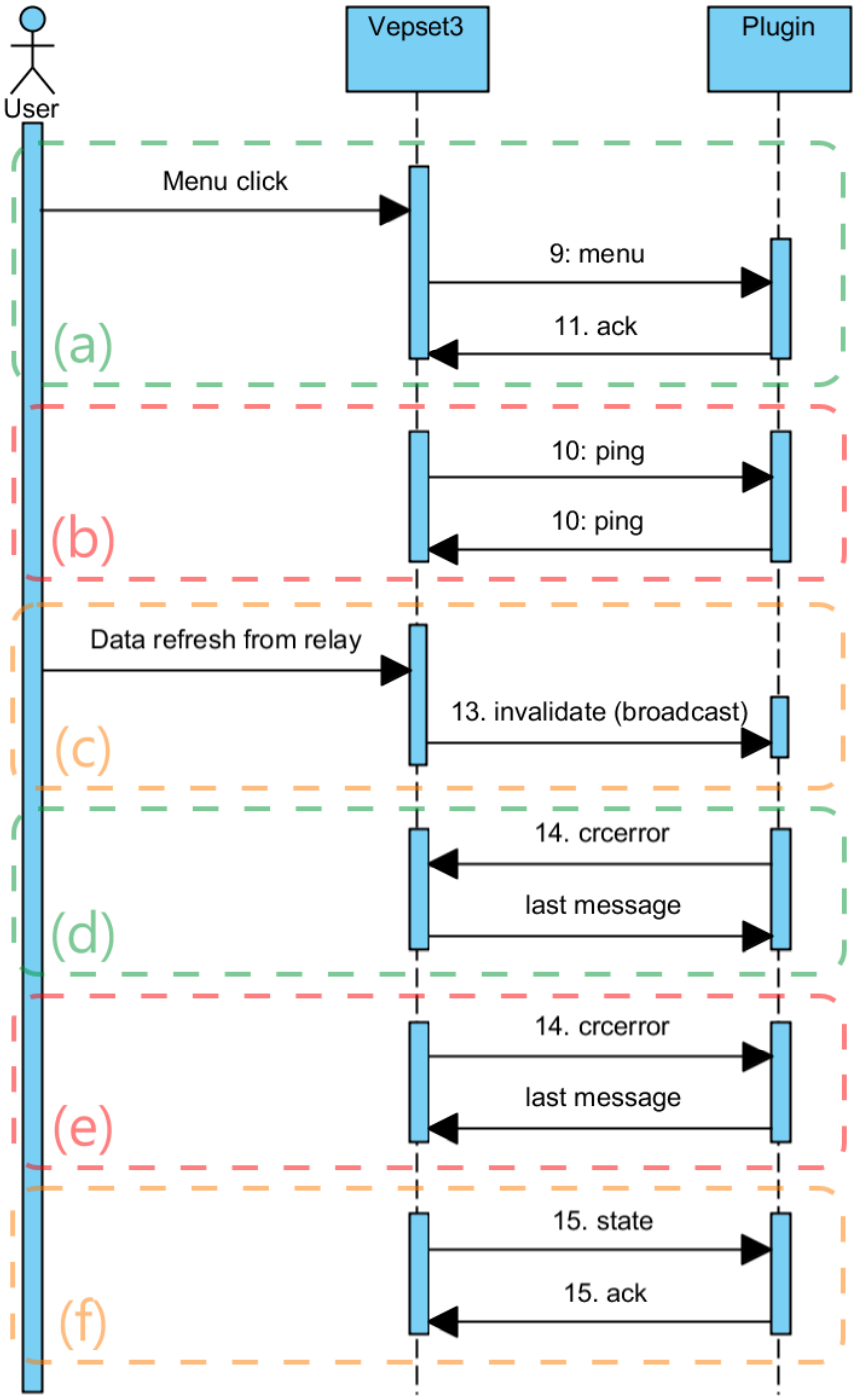

Figure 16. Message exchange diagram of the various messages exchanged between the relay setting tool and the plugins, part 4.

ative to the upper left coordinate of the bitmap. X- and Y-coordinates are represented with 16-bit unsigned integers, by using big-endian encoding. Button is determined by

8-bit unsigned integer following the coordinates. Currently value 0 corresponds to left-click, value 1 means right-click, value 2 is the scrollwheen click, value 3 represents scroll wheen turning up, value 4 is the scroll wheen turning down and value 5 is double-click of the left-button. Plugin must respond to this message with ack and with new bitmap if required. Message exchange shown on Figure 15 case d.

Menu-command is in turn used to inform the plugin about clicks on the plugin's menu displayed on the relay setting tool's menubar. Message-field contains information about the clicked menu. Plugin must respond to this with an ack. Message exchange shown on Figure 16 case a.

When a view is refreshed from a relay in the relay setting tool, the information in the current view might change unexpectedly. Invalidate-command is used to inform the plugins that some view has been refreshed from a relay and the contained information might have become invalid. This is a broadcast-message to all of the plugins and plugins can ignore this if they are not interested on the view in question. Message exchange shown on Figure 16 case c.

CRC error results when the received and calculated checksums do not match. The party which detects the error, sends this message to the other. Correct response to this is to re-send the last message. Message exchange shown on Figure 16 cases d and e.

State-command is sent when the relay setting tool window loses or gains focus, or when the plugin's icon is clicked on the relay setting tool's toolbar. This can be used to hide and active the plugin-window. State-message contains either the number 0 for hiding or number 1 for activating the plugins' window. Message exchange shown on Figure 16 case f.

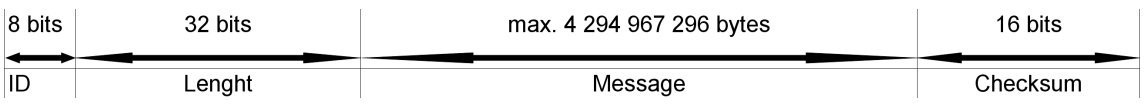

Figure 17. Relay setting tool/plugin communication frame

5.1.4. Message frame

The anatomy of the message frames (inside the TCP/IP-frames) used for the communication is shown on Figure 17. Individual messages sent through the sockets must always begin with the number corresponding to one of the message-types. Following the command ID is the length of the message as 32-bit unsigned integer and after the length-information is the message itself. The message ends with a checksum.

The length is calculated only from the message, not including the command ID, the length-information itself or the checksum. It should also be noted that the length is not for example the total number of UTF-8 characters in the message, but the byte-length of the message. Because of 32-bit unsigned integer is used, a theoretical maximum length of a message is $2^{32} = 4294967296$ bytes. For practical reasons, the individual messages should be limited to few kilobytes.

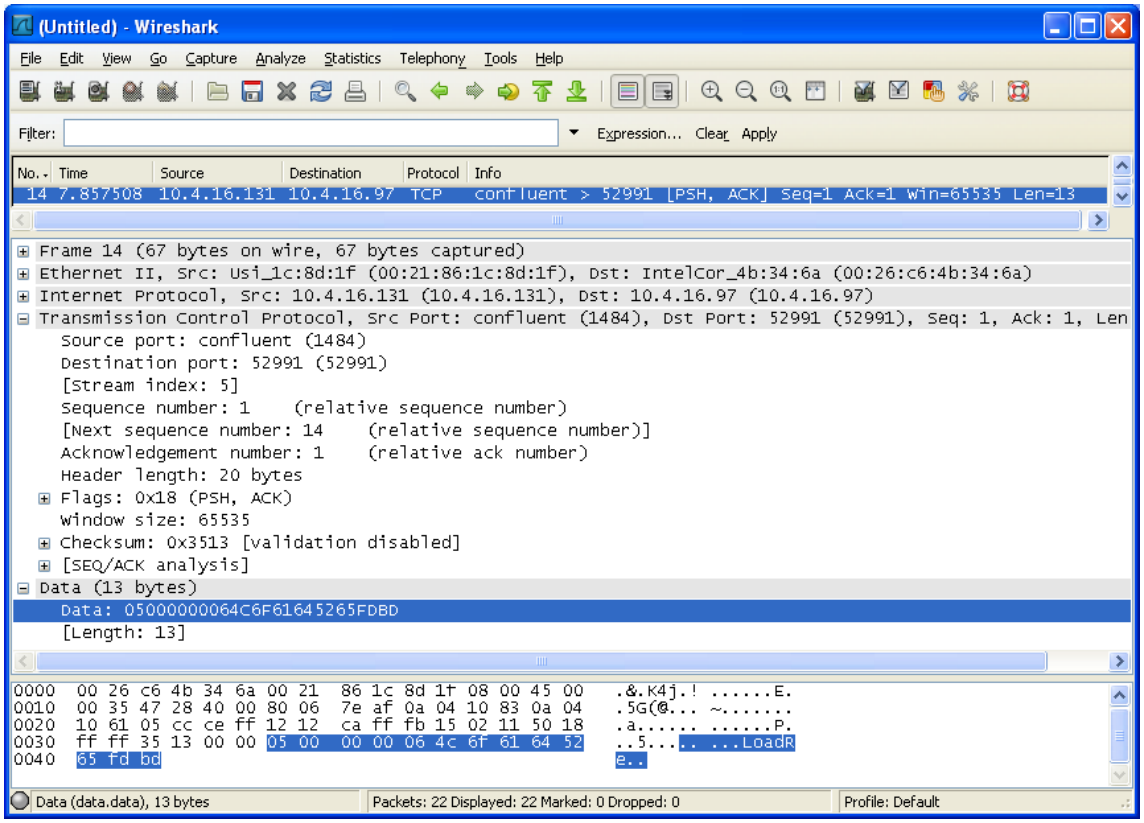

Figure 18. An example of the relay setting tool/plugin communication frame, as seen by Wireshark

Checksum is calculated by first initializing 16-bit unsigned integer to zero. Each byte in the message is then appended to this number. When all of the bytes have been taken into account, the bits on the final product are then inverted.

5.1.5. Example

If for example the original message was an UTF8-text string "LoadRe", a corresponding hexadecimal representation would be 4c:6f:61:64:52:65. In this case the length of the message would be 6 bytes, and this number represented as a 32 bit unsigned integer would be 00:00:00:06. If the used command ID would be 5, then the resulting checksum would be 1111110110111101 (or fd:bd as hexadecimal). A Wireshark-capture of such frame is shown on Figure 18

A more detailed image of the example-message inside the TCP/IP-frame is shown on Figure 19. Here the command ID is shown with a yellow background, lenght of the message is shown with green background, the message itself is shown on turquoise background and the checksum is shown on the pink background.

05:00:00:00:06:4c:6f:61:64:52:65:fd:bd

Figure 19. An example of the relay setting tool/plugin communication frame

5.1.6. Timeouts

When there is no other communication, the relay setting tool sends the ping-messages every 1000 millisecond and plugins have to respond within 1000 milliseconds. If the messages do not arrive within these time-limits, the other end of the communication is considered to be unresponsive and is assumed dead. Plugin should close itself when it notices that the relay setting tool is unresponsive. When the plugin seems to be unresponsive, the relay setting tool in turn tries to close it and start a new instance of the plugin.

Also if an incomplete message is received and no further data is received within 1000 milliseconds, the other end of the communications should be assumed dead.

5.1.7. CRC errors

In addition the checksums calculated and added to the end of each message sent, the received checksums must be validated. When one party (the relay setting tool or a pluging) detects invalid checksum, it must clear the receive-buffer and respond with CRC error -message. When CRC error -message is received, previously sent message must be resent. When three consequent errors are detected, the other party of the communication is no longer considered sane and the communication must be terminated.

5.1.8. Other considerations

Plugins should always be prepared for those cases when they are used for example with some older relay that either only partially supports the required functionalities or does not

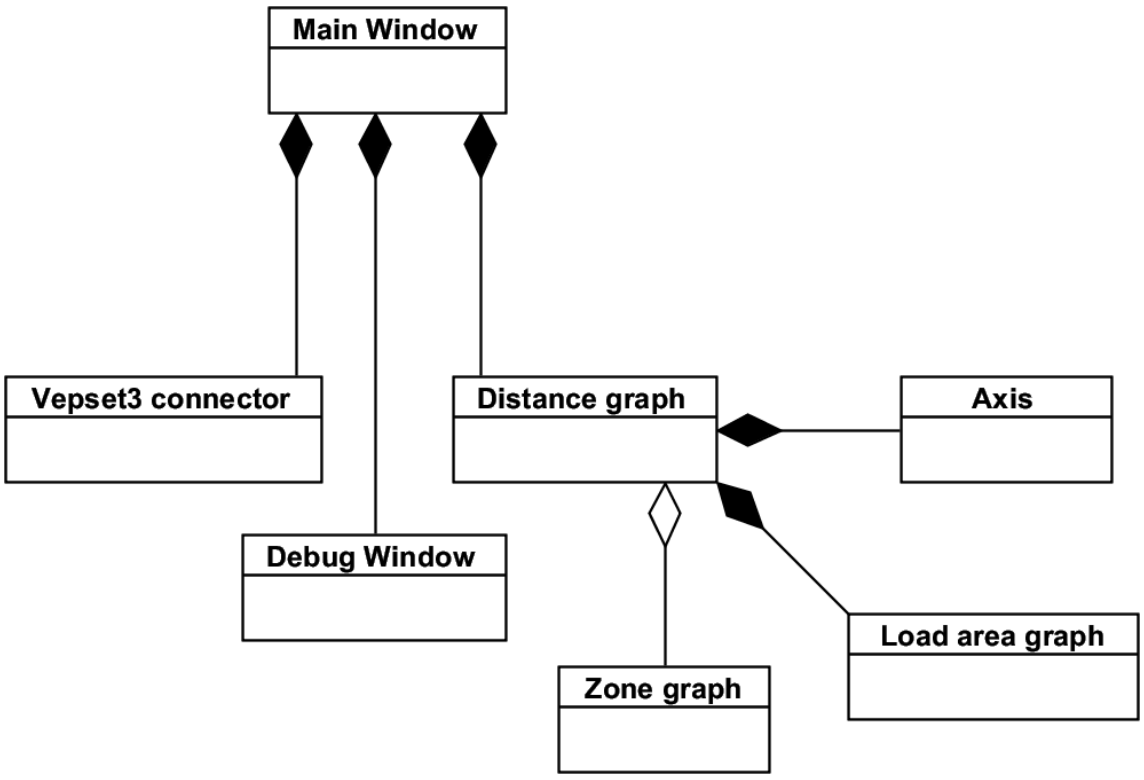

Figure 20. UML-model detailing how the software was divided into smaller modules.

support them at all. Plugins should also take into account the fact that the relay setting tool is the host of the communication and it can close the plugin by killing the process at any given time if the plugin does not communicate properly.

5.2. Drawing the distance protection graph

This section describes the functionality and implementation of the program which draws the distance protection graphs. The structure of the program is shown as an UML-diagram on the Figure 20. The first one to start is the main window. The main window uses the relay setting tool Connector library to establish a connection to the relay setting tool. Simultaneously the main window can start up a debugging window which logs the communication between the plugin and the relay setting tool.

The main window uses a distance graph component to construct the distance protection graph. The image of the graph representing the distance protection function settings consist of three different parts; the axis R and X, load areas and the different protection zones (and the hysteresis area between the upper and lower parts of the protection zone areas). There can be a maximum of five protection zones active simultaneously. Usually they are set so that the first zone is the inner-most zone and the fifth zone is the outer one. The order can however be different and is completely dependent on the users settings. An example of the distance protection graph is shown on Figure 21 (a), the individual components and their relations in the actual program are also shown on the Figure 20.

Hysteresis-area can be defined as infinitely long line residing on origin at -45° angle, extending $\pm 2^\circ$ on each sides. In other words, the hysteresis areas reside at -43°$-$-47° and $+133^\circ - +137^\circ$. The hysteresis area is static and it can not be adjusted by the user.

Load areas are defined with two isosceles triangles which are positioned along the X-axis, peak located in the origin. The angles between the triangle sides and X-axis can be adjusted with the "Load angle"-parameter and the triangles top can be cut off with the "Load resistance"-parameter.

Zones are defined with R- and X setting-values. Zones can also be set to operate in forward, reverse and undirectional modes. In the forward-mode the zone is active only on

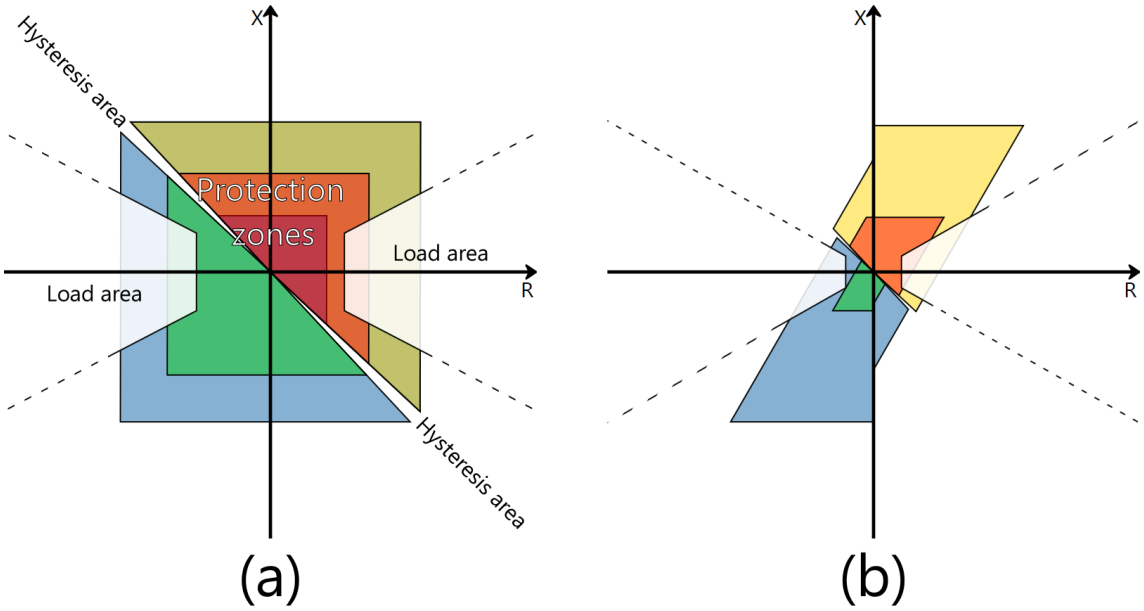

Figure 21. (a) Various parts on the distance protection graph. (b) Effects of the "Line angle"-parameter.

the area above the hysteresis-area and in the reverse-mode the zone is active on the area below the hysteresis-area. In undirectional mode the zone is active on the both sides of the hysteresis-area.

The final parameter which affects the distance protection graph is the "Line angle". The line skews the protection zone polygons. The tilting however does not continue through the X-axis, but stops there. The effects of the skewing are shown on Figure 21 (b).

As the zone areas are almost just ordinary rectangles (as shown on Figure 21 (a)), the first approach to the drawing of the graphs representing the distance protection functions, was to draw the different zones just as rectangles defined by the given R- and X setting-values. On top of the rectangles, would have been drawn the hysteresis-zone and load area. This approach however would have been troublesome to implement in the cases where the zones would have been set to operate in forward- or reverse-modes, because there would have been no way to just draw the other half of the rectangles. Further problems would have arisen on certain "Line angle" values, because these may change the zone areas to something far from a rectangle, as shown on Figure 21 (b).

Because the first method seemed infeasible, another bit more complicated approach was conceived. The basic idea behind this method was that for each of the zone area images a set 7 lines with infinite length were defined based on the various distance protection settings. Equations for these 7 lines would then be used to calculate 17 points of intersection for these lines and the resulting information would be used to construct the polygons defining the distance protection zones. Load area would then finally be drawn on top of the zones.

5.2.1. Calculating the coordinates

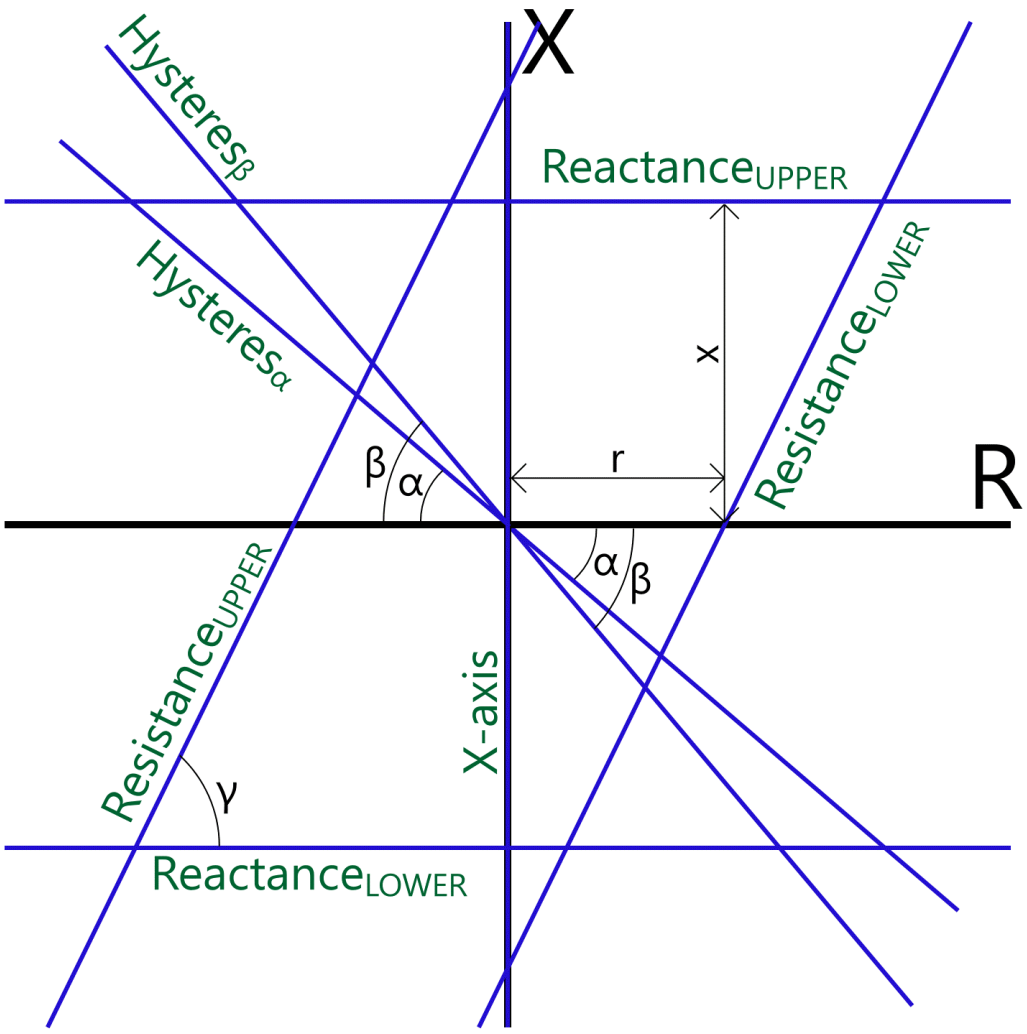

Figure 22. The 7 lines calculated for constructing the distance protection graph.

Figure 22 shows the 7 lines in question along with the names by which they are referred to in this thesis. Here the angles angle α and β define the hysteresis area. The angle α is 43° (45° - 2°) or 0,75rad and the angle β is 47° (45° + 2°) or 0,82rad respectively. Lines $Hysteresis_{\alpha}$ and $Hysteresis_{\beta}$ correspond with these.

The x (reactance) can be set by the user and this defines the lines $Reactance_{UPPER}$ and $Reactance_{LOWER}$. Parameters r (resistance) and the angle γ (which represents the "Line angle"-parameter) can also be modified by the user and these affect the lines $Resistance_{UPPER}$ and $Resistance_{LOWER}$. The final line is the X-axis and this moves only according to the origin.

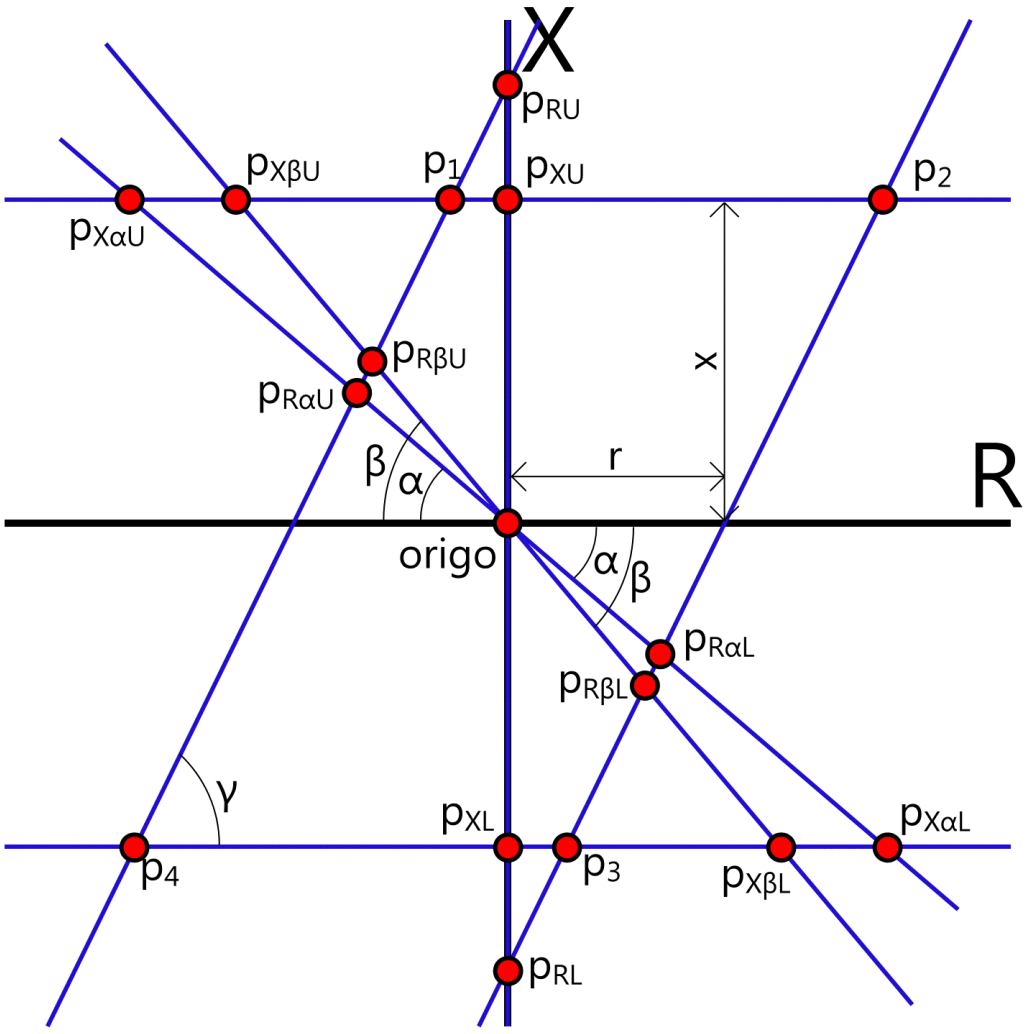

Figure 23. The 17 points calculated for constructing the distance protection graph.

Figure 23 shows the 17 points of intersection for the 7 lines shown on the Figure 22. In computer graphics the origin of an image is usually on the top left corner, so in this case it needs to be moved to the middle of the image, in order to draw the whole graph. Calculating the The shifting of the origin is achieved simply by appending the coordinates of the origin to the coordinates of each of the other points.

As the origin is always known (the coordinates of the center of the image) only 16 points need to be calculated. This number can however be even further decreased by noticing that the upper and lower parts of the distance protection zone graphs are symmetrical

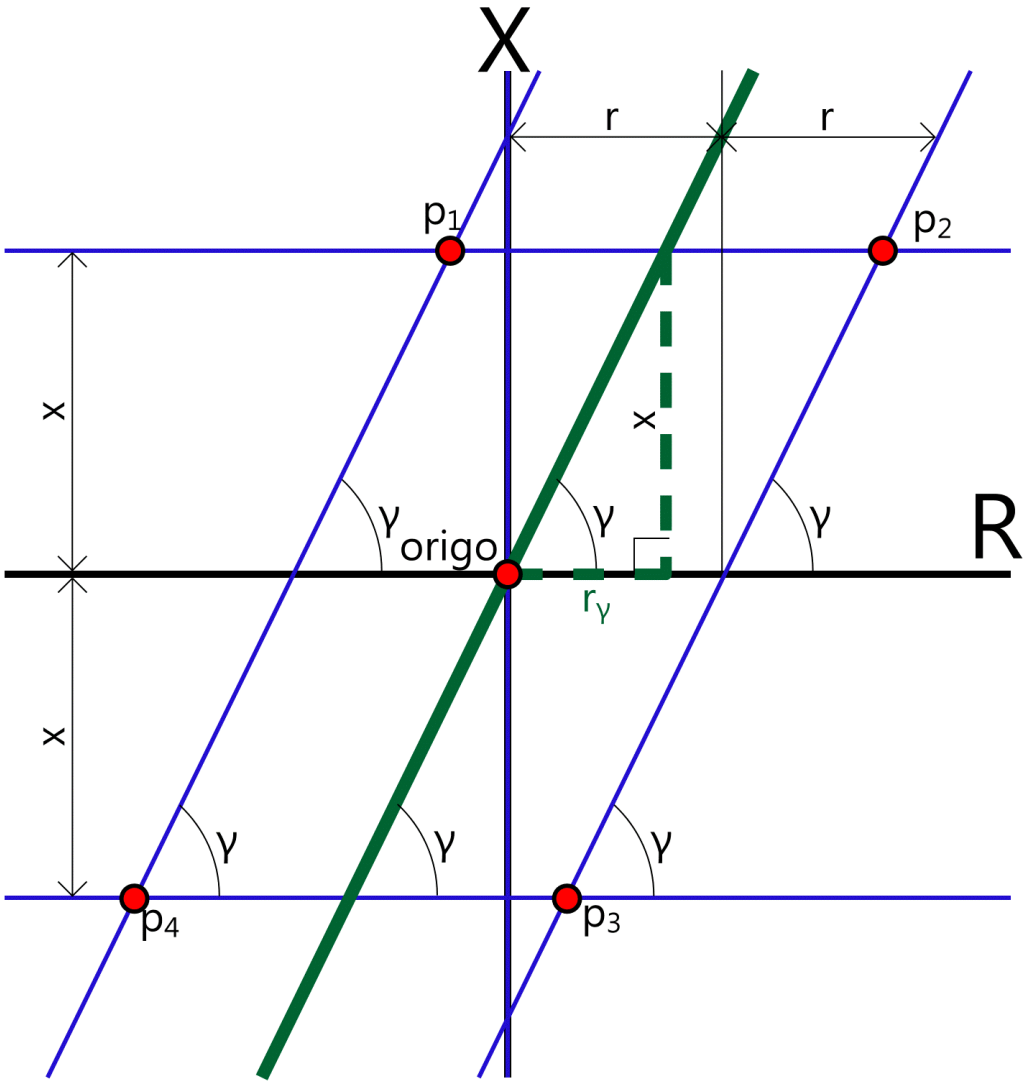

Figure 24. Calculating the coordinates of the points p_1, p_2, p_3 and p_4.

by the R- and X-axis. There is actually no need to calculate each of the 16 points; it is enough to calculate the coordinates for 8 points and the rest can then be derived just by multiplying the other coordinates with -1.

First of all, calculating the coordinates of the points p_{XU} and p_{XU} is really simple, as they can be directly taken from the value x, which in turn is always known. The descriptions and the coordinates for these points are:

— p_{XU}, intersection of $Reactance_{UPPER}$ and $X - axis$

$$p_{XU} = (0, x) \tag{3}$$

— p_{XL}, intersection of $Reactance_{LOWER}$ and $X - axis$

$$p_{XL} = -p_{XU} \tag{4}$$

Calculating the coordinates for the points p_1, p_2, p_3 and p_4 is also pretty straightforward. As it is know from basic trigonometry, tangent is the ratio of the opposite and adjacent sides of a right triangle. If a line parallel to the lines $Reactance_{UPPER}$ and $Reactance_{LOWER}$ (on the Figure 23) is drawn on the origin, a right triangle is formed, where x is the opposite side and r_γ is the adjacent. This is shown as a green triangle on the Figure 24.

Considering how the tangent is defined, in the case of this triangle we can write it in the following way:

$$\tan(\gamma) = \frac{x}{r_\gamma} \tag{5}$$

and from this we can solve the r_γ:

$$r_\gamma = \frac{x}{\tan(\gamma)} \tag{6}$$

and from this the coordinates of the points p_1, p_2, p_3 and p_4 can be easily derived by appending and subtracting the value r. The descriptions and equations for these points are listed below:

— p_1, intersection of $Reactance_{UPPER}$ and $Resistance_{UPPER}$

$$p_1 = (\frac{x}{\tan(\gamma)} - r, x) \tag{7}$$

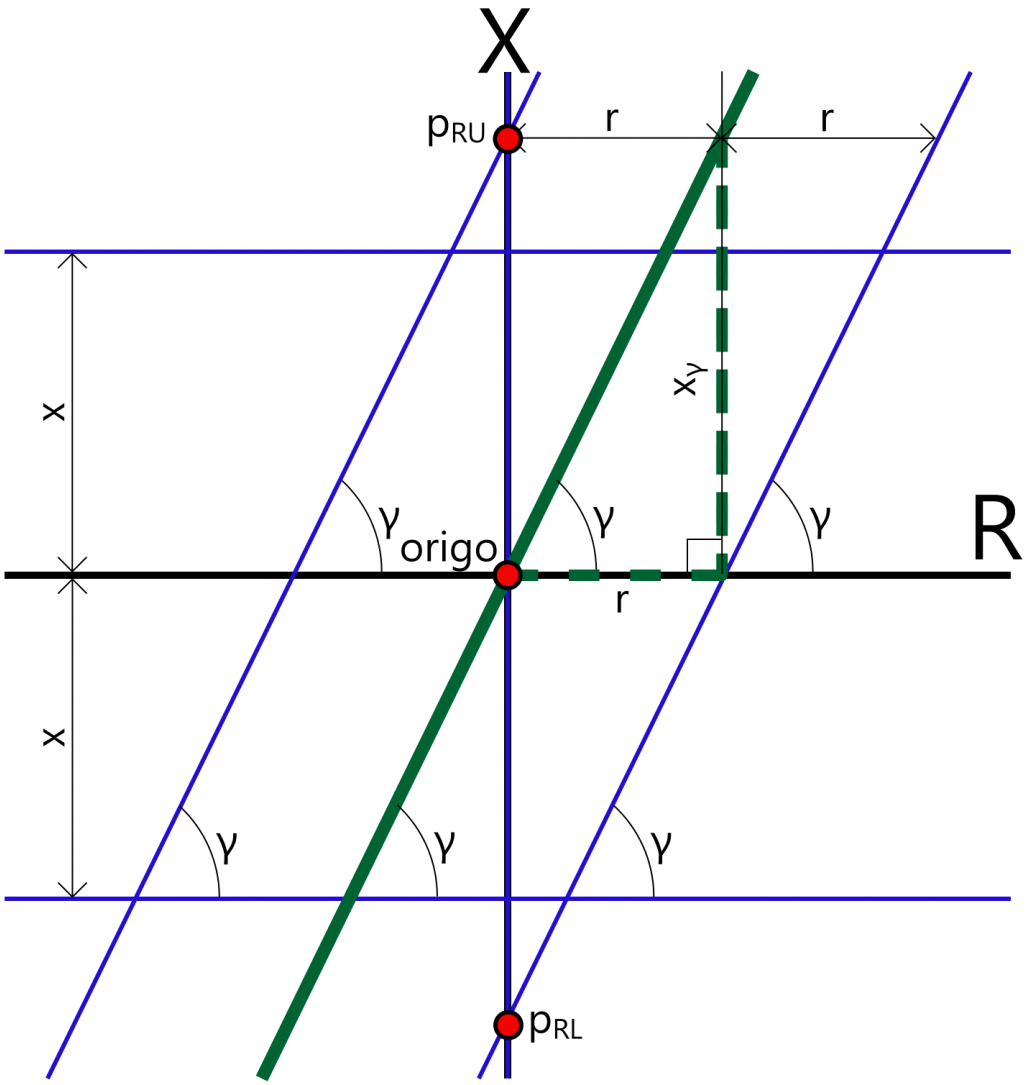

Figure 25. Calculating the coordinates of the points p_{RU} and p_{RL}.

— p_2, intersection of $Reactance_{UPPER}$ and $Resistance_{LOWER}$

$$p_2 = (\frac{x}{\tan(\gamma)} + r, x) \tag{8}$$

— p_3, intersection of $Reactance_{LOWER}$ and $Resistance_{LOWER}$

$$p_3 = -p_1 \tag{9}$$

— p_4, intersection of $Reactance_{LOWER}$ and $Resistance_{UPPER}$

$$p_4 = -p_2 \tag{10}$$

From the previous calculations, the coordinates for the points p_{RU} and p_{RL} can be derived with some relatively simple changes. The triangle on the Figure 24 just needs to be changed so that now the x_γ is the opposite side and r is the adjacent side of the triangle, as shown on the Figure 25. The equation of the angle γ is now

$$\tan(\gamma) = \frac{x_\gamma}{r} \tag{11}$$

and from this the x_γ can be derived as follows

$$x_\gamma = r \tan(\gamma) \tag{12}$$

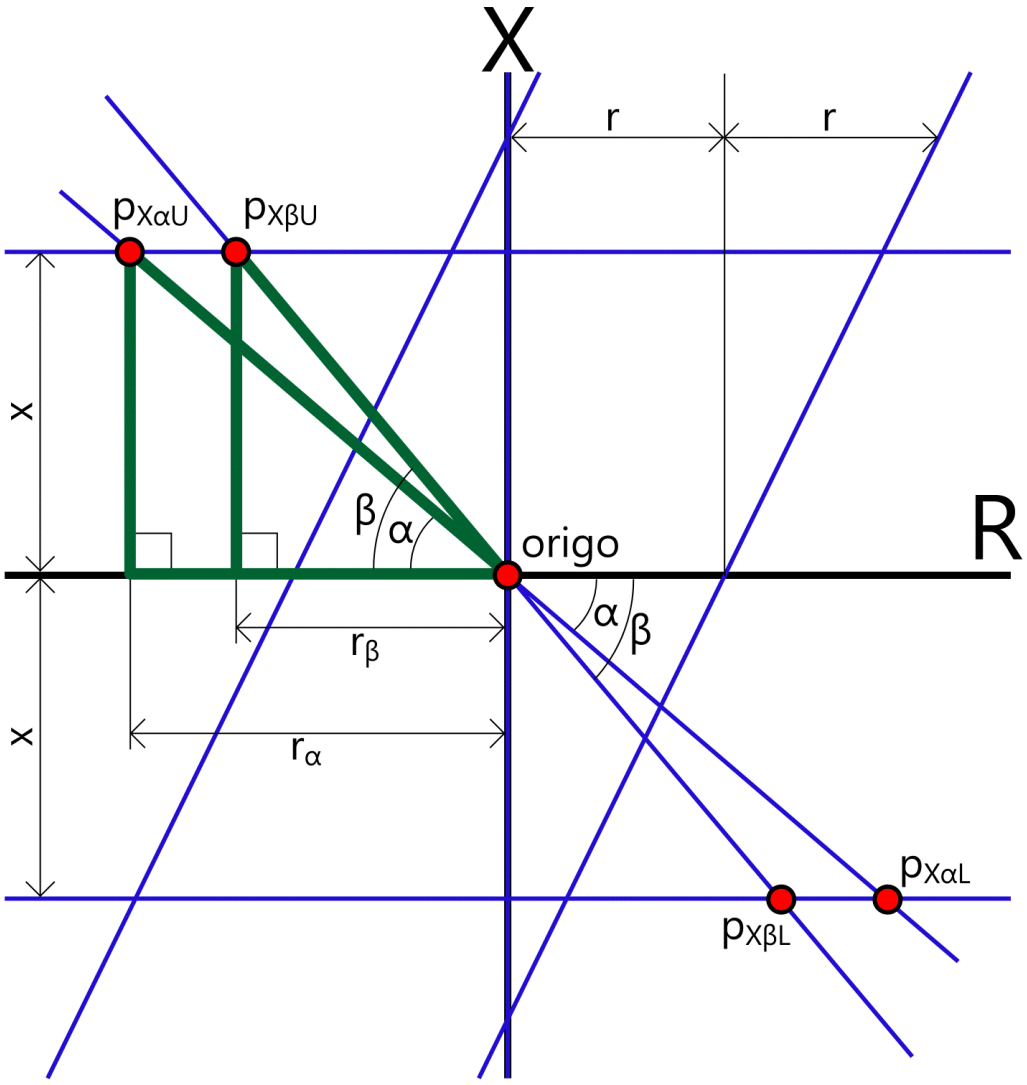

Figure 26. Calculating the coordinates for the points $p_{X\alpha U}$, $p_{X\beta U}$, $p_{X\alpha L}$ and $p_{X\beta L}$.

. The descriptions and equations for these points are listed below:

— p_{RU}, intersection of $Resistance_{UPPER}$ and $X - axis$

$$p_{RU} = (0, r\tan(\gamma)) \tag{13}$$

— p_{RL}, intersection of $Resistance_{LOWER}$ and $X - axis$

$$p_{RL} = -p_{RU} \tag{14}$$

Again, with some minor modifications, the same tangent equation can be applied for calculating the coordinates for the points $p_{X\alpha U}$, $p_{X\beta U}$, $p_{X\alpha L}$ and $p_{X\beta L}$. The triangles for these calculations are shown on the Figure 26. In this case the values r_α and r_β need to be solved and this is done as follows

$$\begin{aligned} \tan(\alpha) &= \frac{x}{r_\alpha} \\ r_\alpha &= \frac{x}{\tan(\alpha)} \end{aligned} \tag{15}$$

$$\begin{aligned} \tan(\beta) &= \frac{x}{r_\beta} \\ r_\beta &= \frac{x}{\tan(\beta)} \end{aligned} \tag{16}$$

and again these functions can be used to calculate the coordinates. The angles α and β are always known, as stated previously in this thesis, so they can be placed directly to the equations. The descriptions of the points and their final equations are listed below:

— $p_{X\alpha U}$, intersection of $Reactance_{UPPER}$ and $Hysteresis_\alpha$

$$p_{X\alpha U} = (-\frac{x}{\tan(43\,^\circ)}, x) \tag{17}$$

— $p_{X\beta U}$, intersection of $Reactance_{UPPER}$ and $Hysteresis_\beta$

$$p_{X\beta U} = (-\frac{x}{\tan(47\,^\circ)}, x) \tag{18}$$

— $p_{X\alpha L}$, intersection of $Reactance_{LOWER}$ and $Hysteresis_{\alpha}$

$$p_{X\alpha L} = -p_{X\alpha U} \tag{19}$$

— $p_{X\beta L}$, intersection of $Reactance_{LOWER}$ and $Hysteresis_{\beta}$

$$p_{X\beta L} = -p_{X\beta U} \tag{20}$$

Finally the only points remaining are $p_{R\alpha U}$, $p_{R\beta U}$, $p_{R\alpha L}$ and $p_{R\beta L}$. These are a bit more tricky, as they lie in the intersection of two angled lines, but luckily due to the symmetrical nature of the points, the equation to only one point needs to be solved as it can be then easily applied to the other.

There are probably several different ways for solving the coordinates for these points, but the approach taken here defines first the equations for the two intersecting lines $Resistance_{UPPER}$ and $Hysteresis_{\alpha}$. It is known from basic algebra, that an equation for a line can be defined as

$$y = kx + b \tag{21}$$

, where k defines the slope of the line and b is the point $(0, b)$ where the line intersects with the y-axis. The slope k is the angle α between the line and x-axis, between $]-90\,°, 90\,°]$ and can also be defined as

$$k = \tan\alpha \tag{22}$$

. The α and x here of course are not the same as previously for example on the Figure 23. To avoid confusion in the following calculations, the line equation should probably be written in a form

$$X = \tan(\gamma)R + b \tag{23}$$

, because the distance protection graph is presented on R/X-axis instead of X/Y-axis.

Following the previously given description for the line equation, the line $Hysteresis_{\alpha}$ can be defined as following way:

$$X = \tan(\alpha)R + 0 \tag{24}$$

, because the line goes through origin and thus $b = 0$. In the case of the line $Resistance_{UPPER}$, the point intersection with the vertical axis (the X-axis in this case) has already been calculated previously in this chapter, it is the same as was the second coordinate of the point p_{RU} and thus the equation for this line is

$$X = \tan(\gamma)R + r\tan(\gamma) \tag{25}$$

Now that the equations for the lines $Resistance_{UPPER}$ and $Hysteresis_{\alpha}$ have been defined, the information can be used to solve the point of intersection for these lines. The solving of this linear system of equations is done as follows

$$\begin{cases} X = \tan(\alpha)R + 0 \\ X = \tan(\gamma)R + r\tan(\gamma) \end{cases} \tag{26}$$

$$\begin{aligned} \tan(\alpha)R &= \tan(\gamma)R + \tan(\gamma)r \\ \tan(\alpha)R - \tan(\gamma)R &= \tan(\gamma)r \\ R(\tan(\alpha) - \tan(\gamma)) &= \tan(\gamma)r \\ R &= r\frac{\tan(\gamma)}{\tan(\alpha) - \tan(\gamma)} \end{aligned} \tag{27}$$

and when the value for the R is calculated, the equation for the second coordinate can be easily solved from the equation of the line $Hysteresis_{\alpha}$:

$$\begin{cases} X &= \tan(\alpha)R + 0 \\ R &= r\frac{\tan(\gamma)}{\tan(\alpha)-\tan(\gamma)} \end{cases} \tag{28}$$

$$X = \tan(\alpha)r\frac{\tan(\gamma)}{\tan(\alpha) - \tan(\gamma)} \tag{29}$$

The angles α and β are again constant so their values can be directly added to the equations. The following list gives the descriptions and final equations for the remaining four points:

— $p_{R\alpha U}$, intersection of $Resistance_{UPPER}$ and $Hysteresis_{\alpha}$

$$p_{R\alpha U} = (r\frac{\tan(\gamma)}{\tan(-43\,^{\circ}) - \tan(\gamma)}, \tan(43) r\frac{\tan(\gamma)}{\tan(-43\,^{\circ}) - \tan(\gamma)}) \tag{30}$$

— $p_{R\beta U}$, intersection of $Resistance_{UPPER}$ and $Hysteresis_{\beta}$

$$p_{R\alpha U} = (r\frac{\tan(\gamma)}{\tan(-47\,^{\circ}) - \tan(\gamma)}, \tan(47) r\frac{\tan(\gamma)}{\tan(-47\,^{\circ}) - \tan(\gamma)}) \tag{31}$$

— $p_{R\alpha L}$, intersection of $Resistance_{LOWER}$ and $Hysteresis_{\alpha}$

$$p_{R\alpha L} = -p_{R\alpha U} \tag{32}$$

— $p_{R\beta L}$, intersection of $Resistance_{LOWER}$ and $Hysteresis_{\beta}$

$$p_{R\beta L} = -p_{R\beta U} \tag{33}$$

5.2.2. Constructing the polygon

When coordinates for all of the points have been acquired, the next step is to use the information to construct the protection zone polygons. Figure 27 shows an example of different kinds of protection zone graphs which are the results of different setting parameters. Part (a) shows a graph where the set reactance (x) is greater than the set resistance (r) and the line angle is 90°. On the other hand, the part (b) show the graph which results when the set resistance (r) is greater than the set reactance (x), while the line angle still being 90°. In both of these cases the number of vertices on each of the polygons (upper and lower parts of the graph) is 5. On the first case, the points which define the upper part of the graph are $origin$, $p_{R\alpha U}$, p_1, p_2 and $p_{R\alpha L}$, the lower part of the graph is defined by the points symmetrical to these. However in the case of the graph shown on the part (b) of the Figure 27 the graph is defined by a different set of points, they being $origin$, $p_{X\beta U}$, p_2, p_3 and $p_{X\alpha L}$.

Part (c) of the Figure 27 shows the graph which results when the line angle is 90° and resistance is equal to reactance. In this case the polygon has just four corner points which are $origin$, $p_{X\beta U}$, p_2 and $p_{R\alpha L}$. This kind of graph can of course also result in different line angles, when just the ratio between resistance and reactance is appropriate.

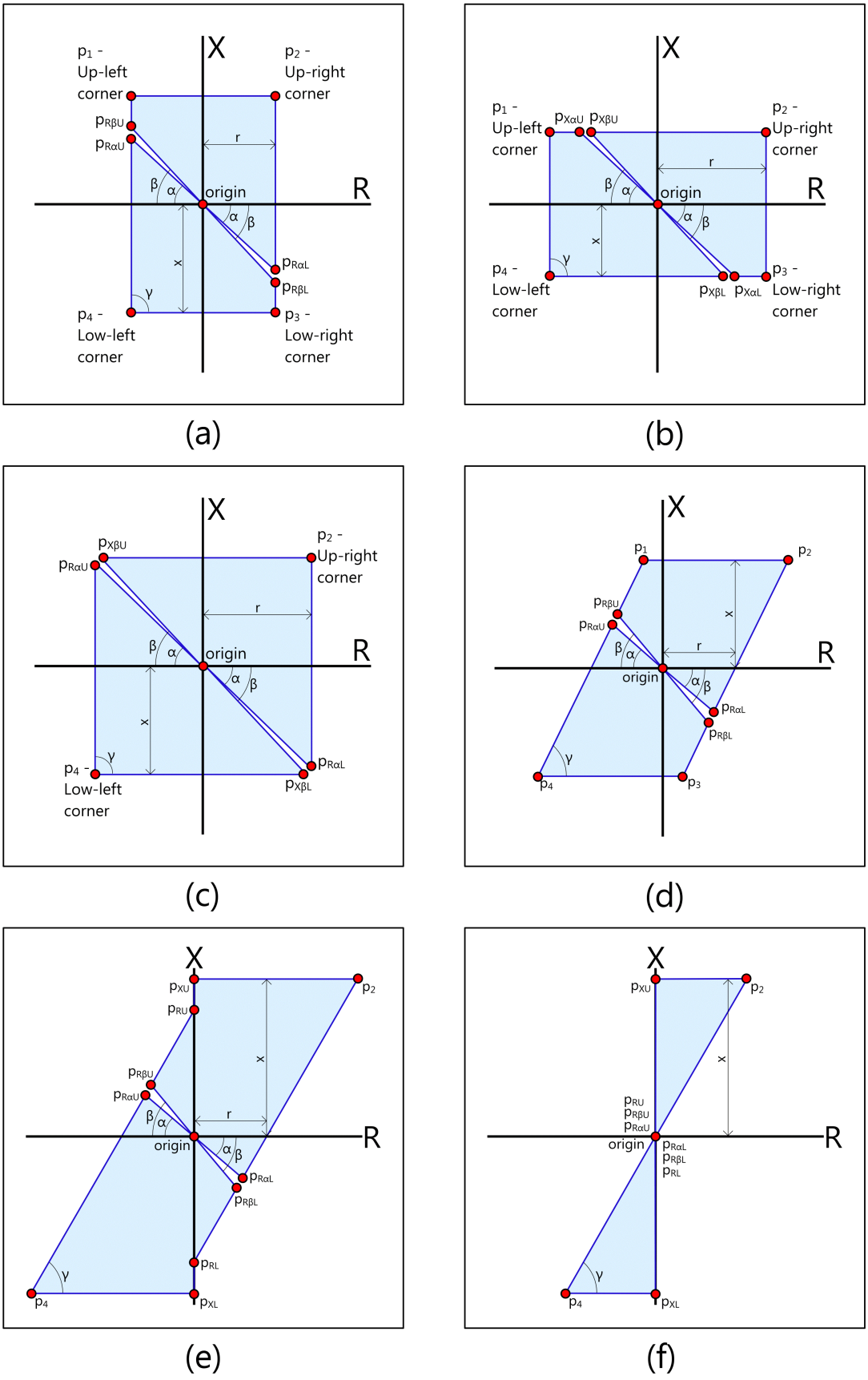

Figure 27. A graph of a polygonal distance protection zone, where (a) line angle = 90° and when $x > r$, (b) line angle = 90° and when $x < r$, (c) line angle = 90° and when $x = r$, (d) line angle < 90° and when $x >> r$, (e) line angle << 90° and when $x >> r$, (f) line angle < 90° and when $x >>> r$

Part (d) of the Figure 27 shows the graph when the line angle is less than 90°. The corner points defining the graph are in this case the same as in part (a). Finally the part (e) shows the graph which results when value set to reactance is increased from the previous case. Now the polygon has 6 corners; the point p_1 is replaced by points p_{RU} and p_{XU}.

Finally the part (f) of the Figure 27 shows the most extreme case, where for example the line angle is smaller than 90°, resistance has been set to minimum and reactance to maximum. Now the polygon seems to have only 3 corners, but the number of corners is actually the same as on part (e). In this case four of the corners are just being rendered to same pixel. If the line ange is set to 90°, the point p_2 will also be rendered to the same pixel as the point p_{XU} and the graph will become just one line on the X-axis.

From the previous description, it can be noted that the only the points $origin$ and p_2 will always be included in the upper polygon. An algorithm needed to be designed in order to determine whether each of the other points should be included. There are obviously many possible ways for implementing it, but one relatively simple version is explained here. This version consists only from a couple of coordinate comparisons.

Again, from the previous description it can be noted that always either the point $p_{R\beta U}$ or $p_{X\beta U}$ is included. From these the one with lower R-coordinate should be included. In other words; the one which is closest to the origin. If the point $p_{R\beta U}$ was included, then the second step is to include either the point p_1 or the two points p_{RU} and p_{XU}. The selection between these can be done by comparing the R-coordinate of the point p_1; p_1 should be included if the coordinate is negative and the other two points should be included if the coordinate was positive.

Final step is to select either the point $p_{R\alpha L}$ or the points p_3 and $p_{X\alpha L}$. The selection between these two can also be done by comparing the values of the R-coordinates of the points $p_{R\alpha L}$ and $p_{X\alpha L}$; the one closest to the origin should be chosen.

Once the corner points of the upper distance protection zone polygon have been found out, the lower polygon is easy to construct just by mirroring the upper one. As the user can set the zone operate in either reverse, forward or undirectional mode, this information is then used to set either the upper, lower or both polygons visible.

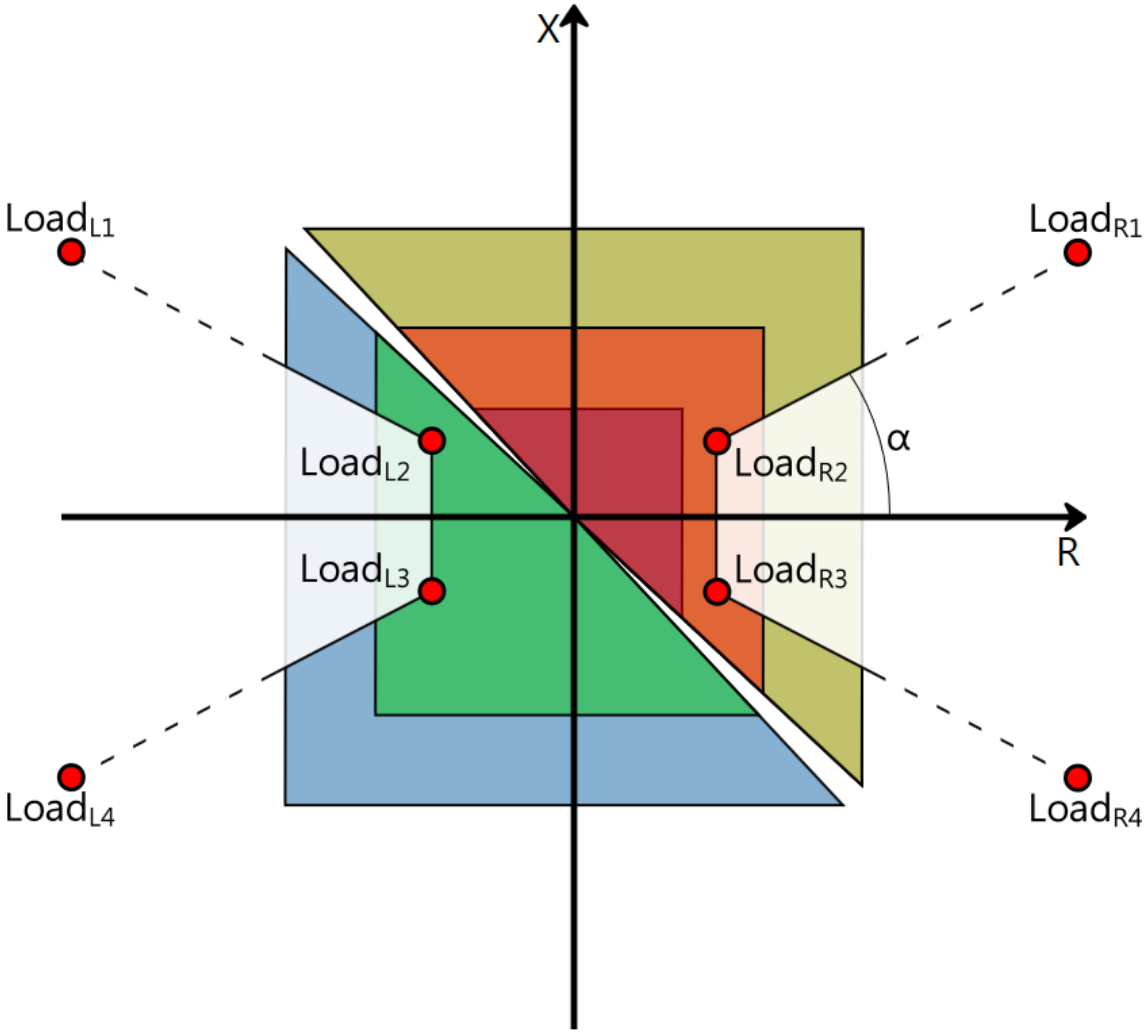

Figure 28. Calculating the points defining the load area polygons.

5.2.3. Load area

There are two load area polygons in the distance protection graph, one polygon on the left and one one the right side. The closed side of the polygons is always facing towards origo and the open side is facing away from the origo. The polygon for the load area is defined by four points which are shown on Figure 28. Again, as it was in the case of the zone area polygons, the areas are symmetrical so only coordinates for two points need to be calculated and the other can be mirrored from the first by multiplying the coordinates with -1 accordingly.

For example in the case of the point $Load_{R2}$, the distance from the origin on the R-axis is

directly set by the user. Angle α is also set by the user and from this the coordinate on the X-axis can be calculated with simple trigonometry. To make one side of the polygon open, the outer points are drawn two pixels outside the image, as the thickness of the border is two pixels. Maximum resistance value is constant 250. Functions for calculating the coordinates of the points are listed below:

$$\begin{aligned} Load_{MaxR} &= Resistance_{Max} + 2 \\ Load_{MaxX} &= \tan(\alpha) * (Resistance_{Max} + 2) \\ Load_{SetR} &= Resistance \\ Load_{SetX} &= \tan(\alpha) * Resistance \end{aligned} \tag{34}$$

$$\begin{aligned} Load_{R1} &= (Load_{MaxR}, Load_{MaxX}) \\ Load_{R2} &= (Load_{SetR}, Load_{SetX}) \\ Load_{R3} &= (Load_{SetR}, -Load_{SetX}) \\ Load_{R4} &= (Load_{MaxR}, -Load_{MaxX}) \end{aligned} \tag{35}$$

$$\begin{aligned} Load_{L1} &= (-Load_{MaxR}, Load_{MaxX}) \\ Load_{L2} &= (-Load_{SetR}, Load_{SetX}) \\ Load_{L3} &= (-Load_{SetR}, -Load_{SetX}) \\ Load_{L4} &= (-Load_{MaxR}, -Load_{MaxX}) \end{aligned} \tag{36}$$

5.2.4. Other features

The distance protection graph drawer which was implemented in this work also has a number of other smaller functionalities which have not yet been described. The first of these is the colouring of the distance protection zone polygons. The user can change the colours of each of the distance zone polygons by clicking them with a mouse. The eight colours available in the program were chosen by using Color Scheme Designer tool[1]. The

[1]http://colorschemedesigner.com, tool by Pert Stanicek, 2010

purpose of this tool is to ease the selection of colour schemes. The colour schemes selected for the work on this thesis are identified by codes `#4f42Pw0w0w0w0` and `#3y42Pw0w0w0w0`. The individual colors were `#3B14AF`, `#FF0700`, `#00C90D`, `#FFD500`, `#0B5FA5`, `#AA00A2`, `#C9F600`, `#C9F600` and they are drawn with 50% transparency in order to show the overlapping zones. Colours are shown on Figure 29.

Figure 29. Colour-selections chosen for the distance protection zone polygons.

Another smaller feature worth describing in more detail is the drawing of the dashed lines on the load area polygons. This is done by taking the leftmost and rightmost lines, $Resistance_{UPPER}$ and $Resistance_{LOWER}$ (shown on the previously mentioned Figure 22), from the biggest distance protection zone polygon. The lines are then duplicated and moved away from the origin on the R-axis, width of the lines is increased and then the lines are drawn on top of the load area polygons with white colour. Another obvious method would have been to calculate where those two lines in question intersect the lines of the load area polygon and then use that information to determine which part should be drawn with solid lines and which part with dashed line.

6. TESTING

This chapter describes the testing processes which were used during the development of the software described in this thesis. First part of this chapter deals with the usability testing which was performed during the user interface designing process. Second part of this chapter deals with the testing done during the application logic development, the traditional software engineering part of the application development process.

6.1. Heuristic evaluation and prototyping

As described previously in chapter 4, the first part of the development process was to design the user interface and a prototyping process was used for this. A prototype of the user interface was designed and the implemented as a functional computer program. A heuristic evaluation was performed on the prototype by giving it to an expert for evaluating. The expert was instructed to try out the prototypes, estimate how the program would ultimate be used, evaluate how the program suited for the purpose and report any and all

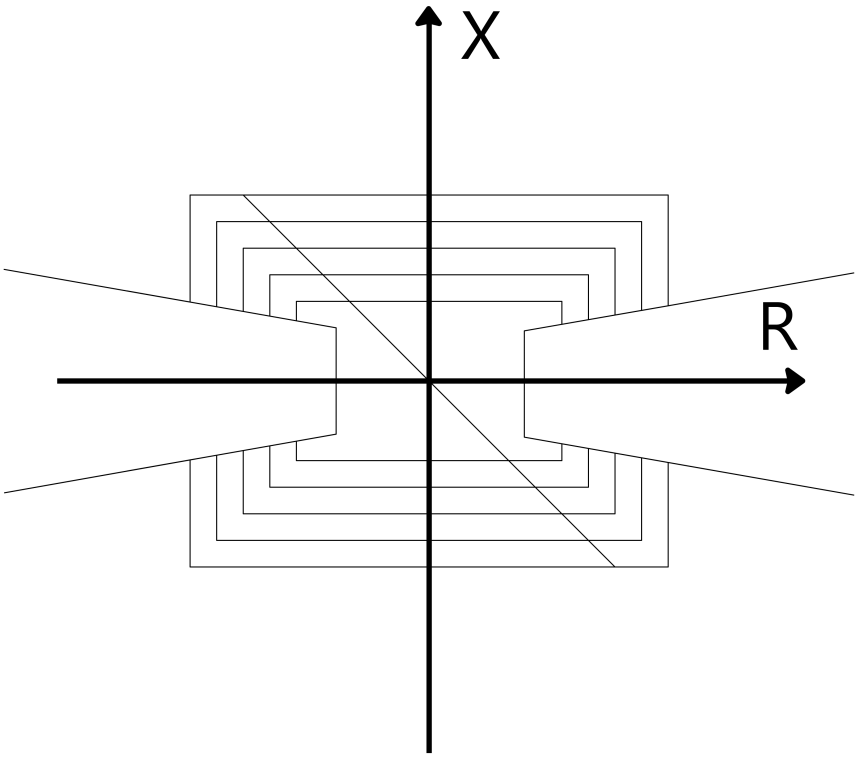

Figure 30. The first prototype of the distance graph drawer.

errors and ideas for improvement. The experts were also observed as they were using the prototype, the observations were then used to determine how the program might be used and this information was subsequently used for improving the program.

Services of several different experts were utilized. After one expert had finished with the analysis, the requirement specification of the program was modified to accommodate the changes and then the program a new prototype was designed and implemented according to the new specification. The modified prototype was given back to the expert for verifying that the changes had been implemented as intended. After the modifications had been accepted, the program was given to another expert for a new heuristic analysis and the cycle was repeated until all of the available experts were satisfied with the program.

The first prototype of the user interface is shown on Figure 30. One of the experts im-

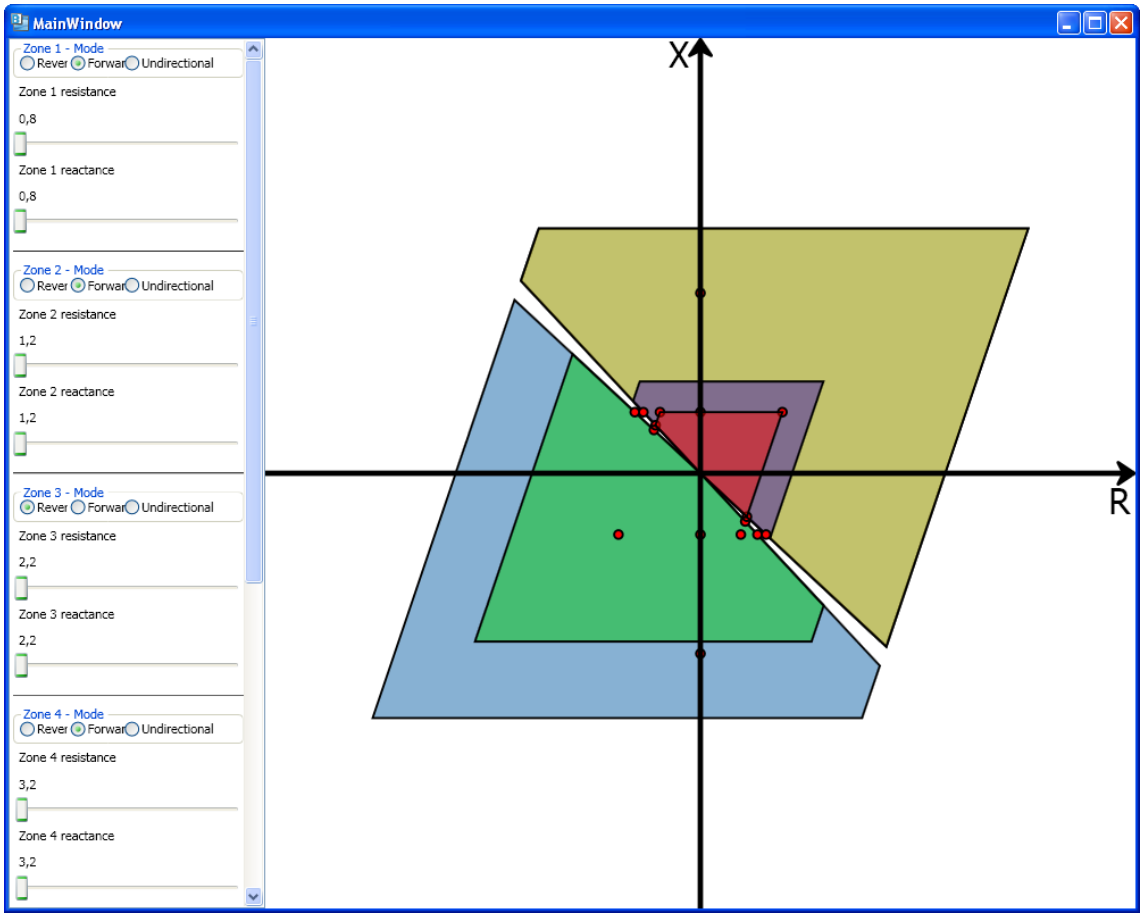

Figure 31. Distance graph user interface prototype and module testing tool.

mediately noted that hysteresis area, which was represented just as a single line in the reference-documentation, should actually be represented as 4° wide cut-out. It was also immediately noted that the graph would be much clearer with some colouring instead of using just transparent rectangles. As the colouring was implemented it was again noted that some of the polygons were overlapping and thus some zone protection area polygons were not visible at all with certain configuration values. The polygons were then made transparent with and additional option for changing the colour by clicking them with a mouse.

The testing also revealed that as the scaling of the graph was permanently set so that it was able to show the largest possible graph, the smaller graphs were too tiny to be of any use. Thus a feature was implemented to automatically scale the graph to fit to the configured parameters. Finally one of the experts noted that the parts of the load area polygon, which were not overlapping with any of the protection zone polygons, should be represented as dashed lines. A somewhat final version of the distance graph drawer has been shown already multiple times on this thesis, for example on the Figures 21 and 31. One of the later prototypes of the user interface is also shown on Figure 31. This prototype was also used during the application logic development and finally during the module testing.

6.2. Module- and integration testing

This section describes the software testing which was done during the application logic development process. A non-formal exploratory white box testing was routinely run by the programmers during the development of the program, but a series of more formal and planned tests were also performed. Several different tools were also developed for aiding both the formal module testing and the routine white box testing, these tools are also described in this section.

Because the software developed here was relatively simple and partially already tested by the heuristic testing and quite comprehensive routine white box testing had already been performed thanks to the developed testing tools, it was decided that complete formal and planned module testing would not done. It should also be noted that the relay setting tool had already been tested on numerous occasions and it had in fact already been in use for

many year, which diminished the need of complete testing on that part. So to be precise, the only part of the program which needed full testing was supporting the plugins and the related interfaces.

Even though a complete module testing was not implemented, neither was the Big Bang integration used. The integration testing was done in two steps. First the software was treated as two modules where the first was the relay setting tool and the other module was the software responsible of generating the distance protection graphs. Second phase of the testing, which was also the final testing phase, was done with the complete system.

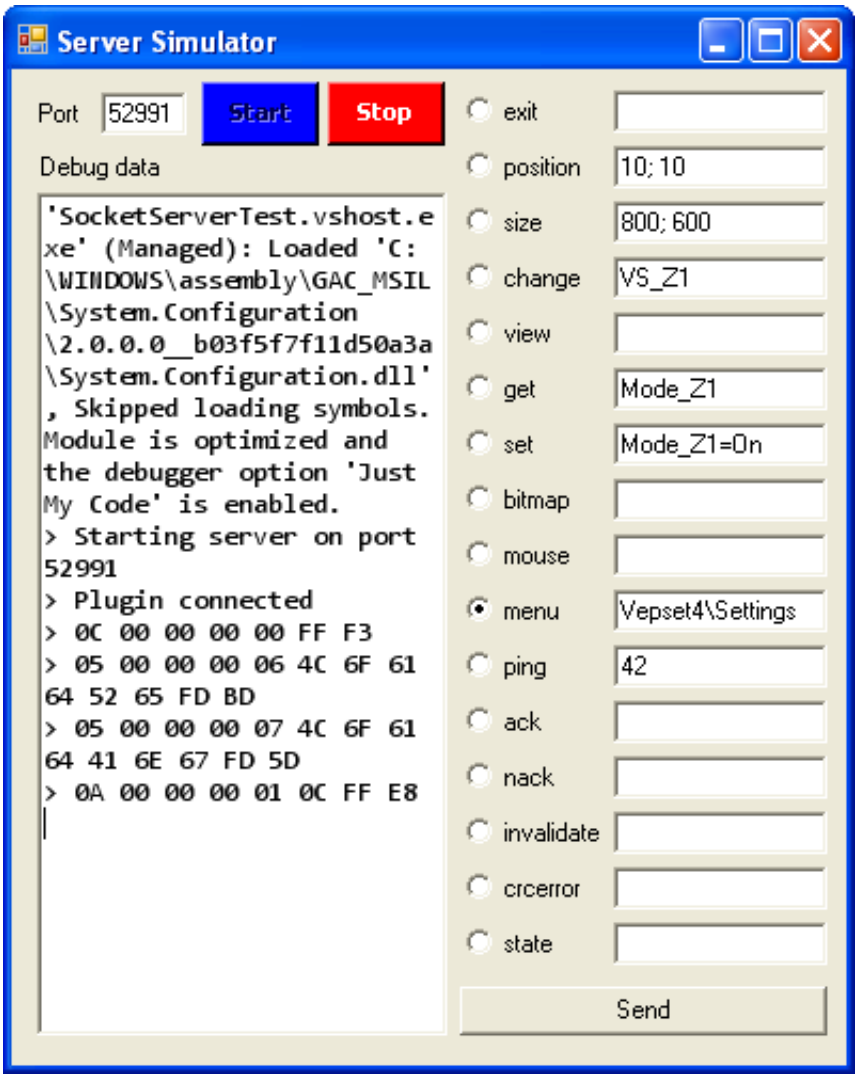

Figure 32. Plugin testing tool, server side.

Chapter 5 of this thesis gives a complete description about the inputs which each part of the program must be able to accept and also how it should react to them. Those descriptions were used as a basis for designing the tests. Some features of the communication protocol between the relay setting tool and plugin were reserved for future implementations and not yet supported. Those features were not tested.

6.2.1. Testing tools

The first testing tool was developed for the distance protection graph, the user interface prototype served as a basis for this tool. This tool was used firstly used as a white box testing tool by the programmer while implementing the module responsible of drawing the distance protection graph. Image of this tool is show on the Figure 31.

This tool allowed the testers to change the graph's input parameters at will and observe

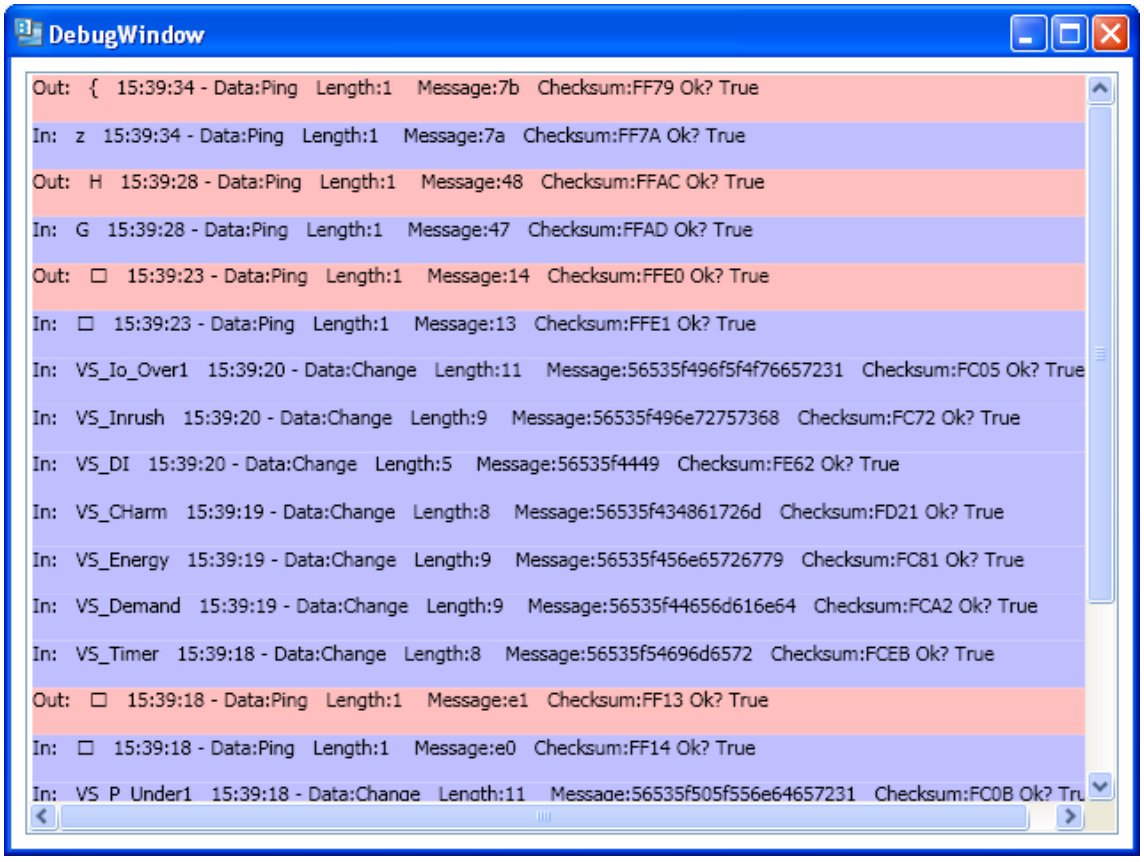

Figure 33. Plugin testing tool, client side.

the changes they caused to the distance protection graph. New graph was generated immediately after changes had been made. The tool was also able to show the points and lines related to constructing the zone graphs, as described previously in this thesis.

The second tool developed for testing purposes is shown on the Figure 32. The purpose of this tool was to simulate the functionalities of the relay setting tool and it was able to send all the messages as the actual relay setting would. The available messages and detailed message exchange diagrams were previously described in more detail in the chapter 5. This tool was used in the integration testing of the graph drawing program.

The third and final testing tool was a simulator of the plugin, shown on the Figure 33. This was primarily developed as a white box testing tool for the person responsible of implementing support for the plugins to the relay setting tool. This tool was able to simulate the functionalities of the plugins. It responded correctly to the messages, as specified previously in the chapter 5 and it was also able to load the configurations from file.

Table 2. Test cases used for testing the user input handling of the plugin handler on the relay setting tool.

Equivalence class	Chosen test case	Correct behavior
Closing program	Close the program	Send exit-message to plugins where: data: 0, length: 0, message: <null> and checksum: FFFF
Changing view	Change view to "DEVICE INFO"	Send change-message to plugins where: data: 3, length: 13, message: 56535f446576696365496e666f and checksum: FB0B
Changing value	Change value "Device name" to "Test"	Send set-message to plugins where: data: 6, length: 15, message: 4465766963654e616d653d54657374 and checksum: FA3C
Clicking menu	Click menu Plugins->Vepset4->Settings	Send menu-message to plugins where: data: 9, length: 38, message: 53657474696e67732f566570736574 332f4d656e752f566570736574342f5365 7474696e6773 and checksum: F188

Table 3. Test cases used for testing the plugin input handling of the plugin handler on the relay setting tool.

Equivalence class	Chosen test case	Correct behavior
Get-query	Send valid get-query where: data: 6, length: 10, message: 4465766963 654e616d65 and checksum: FC1F	Respond with set-message where: data: 6, length: 15, message: 4465766963654e61 6d653d54657374 and checksum: FA3C
Invalid get-query	Send invalid get-query where: data: 6, length: 11, message: 4465766963 65204e616d65 and checksum: FBFE	Respond with nack-message: data: 12, length: 0, message: <null> and checksum: FFF3
Ping-message	Send valid ping-response, where: data: 10, length: 1, message: FF and checksum: FEF5	Do nothing
	Send valid ping-response, where: data: 10, length: 1, message: 00 and checksum: FFF4	Do nothing
Invalid ping-message	Send invalid ping-response where: data: 10, length: 1, message: <anything but previously received + 1> and checksum <valid>	Respond with exit-message and close socket: data: 0, length: 0, message: <null> and checksum: FFFF
Delayed ping-message	Delay the ping-response over 1000ms	Respond with exit-message and close socket: data: 0, length: 0, message: <null> and checksum: FFFF
Ack-message	Send valid ack-message where: data: 15, length: 0, message: <null> and checksum: FFF0	Do nothing
Nack-message	Send valid ack-message where: data: 12, length: 0, message: <null> and checksum: FFF3	Do nothing
Crcerror-message	Send valid crcerror-message where: data: 14, length: 0, message: <null> and checksum: FFF1	Respond with previously sent message
CRC-error in message	Send message with invalid CRC where: data: 15, length: 0, message: <null> and checksum: FFFF	Respond with crcerror-message where: data: 14, length: 0, message: <null> and checksum: FFF1
Invalid message ID	Invalid message ID where: data: 16, length: 0, message: <null> and checksum: FFFF	Do nothing, ignore message
	Invalid message ID where: data: FF, length: 0, message: <null> and checksum: FFFF	Do nothing, ignore message

Table 4. Test cases used for testing the plugin handler's capability of reading input from file.

Equivalence class	Chosen test case	Correct behavior
Wrong filetype	640 randomly chosen characters saved as Settings.xml	Abord loading the file, do nothing
Empty file	Empty file saved as Settings.xml	Abord loading the file, do nothing
Malformed file	Example-file shown on Chapter 5.1.2, first and last rows removed	Load as much as possible and use defaults on those which could not be loaded, abort if loading was not possible at all
	Example-file shown on Chapter 5.1.2, all ending tags removed	Load as much as possible and use defaults on those which could not be loaded, abort if loading was not possible at all
Valid file	Example-file shown on Chapter 5.1.2	Load file, start plugin
Invalid Location-field	Example-file shown on Chapter 5.1.2, last five characters removed from "Location"-field	Abort loading the file, do nothing
Valid Port-field, Port = 1-65535	Example-file, modify value of "Port"-field to 1	Load file, use selected port if possible
	Example-file, modify value of "Port"-field to 65535	Load file, use selected port if possible
Invalid Port-field, Port<=0	Modify value of "Port"-field to 0	Load file, choose port automatically
	Modify value of "Port"-field to -1	Load file, choose port automatically
Invalid Port-field, Port>65535	Modify value of "Port"-field to 65536	Load file, choose port automatically
Invalid Port-field, characters	Set value of "Port"-field to character 'a'	Load file, choose port automatically
Valid AutoStart-field, true	Set AutoStart=true	Load file, start plugin
Valid AutoStart-field, false	Set AutoStart=false	Load file, don't start plugin
Invalid AutoStart-field	Set AutoStart=abc	Load file, don't start plugin
Valid position-field	Set Position=(0,0)	Load file, start plugin normally
	Set Position=(800,600)	Load file, start plugin normally
Invalid position-field, <0	Set Position=(-1,-1)	Load file, start plugin at coordinates 0,0
Invalid position-field, > screen resolution	Set Position=(1681,1051)	Load file, start plugin at coordinates 0,0

Continues on the following page...

...Table 4 continues from previous page.

Equivalence class	Chosen test case	Correct behavior
Invalid position-field, characters	Set Position=(a,a)	Load file, start plugin at coordinates 0,0
Valid size	Set Size=(1,1)	Load file, start plugin normally
Invalid size <0	Set Size=(-1,-1)	Load file, start plugin with size 800, 600
Invalid size, characters	Set Size=(a,a)	Load file, start plugin with size 800, 600
Valid mode	Set Mode=Window	Load file, start plugin normally
	Set Mode=Embedded	Load file, start plugin normally
	Set Mode=Hidden	Load file, start plugin normally
Invalid mode	Set Mode=-1	Load file, start plugin in Window-mode

Table 5. Test cases used for testing the user inputs on the plugin.

Equivalence class	Chosen test case	Correct behavior
Closing program	Close the program	Program closes
Resizing window	Resize window	Graph scales automatically to fit the new window size
Clicking the zone polygons	Click zone-polygons with mouse	The clicked polygons change colour
Invalid input	Press keys	Do nothing, ignore input

6.2.2. Testing the relay setting tool

The testing of the relay setting tool was done as a black box testing, because the person responsible for the testing was not the same as the person who implemented the program. The software testing technique known as equivalence partitioning was also used here.

The part of the program which was being tested here was able to accept data from three different input sources. The first was the data being fed by the user to the program, the second source of input was the plugin program and third source was data read from file. These were the first three major equivalence classes. As these classes further encompass different types of inputs which are processes in different manner, these classes needed to be divided into smaller classes based on the individual input parameters. The individual input parameters again usually contained several different equivalence classes, at least for the valid and invalid inputs. Finally a couple of individuals from each equivalence classes

Table 6. Test cases used for testing the data input from the relay setting tool to the plugin.

Equivalence class	Chosen test case	Correct behavior
Exit-message	Send exit-message: data :0, length: 0, message: <null> and checksum: FFFF	Plugin closes
Ping-message	Send valid ping-response, where: data: 10, length: 1, message: FF and checksum: FEF5	Respond with ping-message where: data: 10, length: 1, message: 00 and checksum: FFF4
Delayed ping-message	Pause the sending of ping-messages for over 1000ms	Plugin closes
Ack-message	Send valid ack-message where: data: 15, length: 0, message: <null> and checksum: FFF0	Do nothing
Nack-message	Send valid ack-message where: data: 12, length: 0, message: <null> and checksum: FFF3	Do nothing
Crcerror-message	Send valid crcerror-message where: data: 14, length: 0, message: <null> and checksum: FFF1	Respond with previously sent message
CRC-error in message	Send message with invalid CRC where: data: 15, length: 0, message: <null> and checksum: FFFF	Respond with crcerror-message where: data:14, length:0, message:<null> and checksum:FFF1
Invalid message ID	Send invalid where: data: 16, length: 0, message: <null> and checksum: FFFF	Do nothing, ignore message
	Send invalid message where: data: FF, length: 0, message: <null> and checksum: FFFF	Do nothing, ignore message
Valid change message	Send change VS_Zcomm message: data: 3, length: 8, message: 56535f5a436f6d6d and checksum: FD06	Respond with a sequence of get-message queries with the following messages: LoadRe, LoadAng, LineAngle, Mode_Z1, Z1X, Z1R, Enable_Z1, Mode_Z2, Z2X, Z2R, Enable_Z2, Mode_Z3, Z3X, Z3R, Enable_Z3, Mode_Z4, Z4R, Z4X, Enable_Z4, Mode_Z5, Z5X, Z5R, Enable_Z5
Invalid change message	Send change VS_DeviceInfo message: data: 3, length: 13, message: 56535f44657669 6365496e666f and checksum: FB0B	Do nothing

Continues on the following page...

...Table 6 continues from previous page.

Equivalence class	Chosen test case	Correct behavior
Valid set-message	Send set-messages for each of the processed parameters: LoadRe, LoadAng, LineAngle, Mode_Z1, Z1X, Z1R, Enable_Z1, Mode_Z2, Z2X, Z2R, Enable_Z2, Mode_Z3, Z3X, Z3R, Enable_Z3, Mode_Z4, Z4R, Z4X, Enable_Z4, Mode_Z5, Z5X, Z5R, Enable_Z5. Full description of the valid messages omitted here.	Change image accordingly and respond with ack-message: data: 15, length: 0, message: <null> and checksum: FFF0
Invalid set-message	Send set-message for parameter DeviceName, data: 6, length: 15, message: 4465766963654e61 6d653d54657374 and checksum: FA3C	Do nothing

were chosen as the atomic test cases. Principles of boundary testing were used in the selection of the test cases.

The equivalence classes, test cases and expected responces are listen on the Table 2. Table 3 lists the test cases concerning the input from the plugins and finally Table 4 lists the test cases about reading data from file. Tests were run by using two plugins to be sure that the program supported multiple plugins.

6.3. Testing the plugin

Similar tests were designed and performed for the plugin as was done with the plugin handler module on the relay setting tool. As it was the case with the plugin handler, the plugin can also receive input from three different sources: from user, the communication from the relay setting tool and finally reading data from file. There is however one additional source of input, which is the command-line argument the relay setting tool gives to the plugin when the plugin is started. This time also the source code of the software was examined in order to determine good test cases and thus this was white box testing.

The test cases shown on Table 4 can be used with the plugins for testing the loading data from the file, new test cases needed to be designed for the other inputs. Cases used for

Table 7. The equivalent classes and test cases used for the final integration testing.

Closing the plugin	Close the plugin-window	Plugin closes
Resizing the plugin window	Resize window	Graph scales automatically to fit the new window size
Changing the graph colours	Click zone-polygons with mouse	The clicked polygons change colour
Invalid input on the graph	Activate graph, press keys on keyboard	Do nothing, ignore input
Activating graph	Click on the "DISTANCE COMMON SETTINGS"-menu	Graph is shown
Closing the plugin	Close the plugin-window	Plugin closes
Resizing the plugin window	Resize window	Graph scales automatically to fit the new window size
Changing the graph colours	Click zone-polygons with mouse	The clicked polygons change colour
Invalid input on the graph	Activate graph, press keys on keyboard	Do nothing, ignore input
Activating graph	Click on the "DISTANCE COMMON SETTINGS"-menu	Graph is shown
Deactivate graph	Click on the "DEVICE INFO"-menu	Graph is hidden
Input Load setting R, valid range 0.05-250	Set to value 0.05	Produce graph where load area polygon value R=0.05
	Set to value 250	Produce graph where load area polygon value R=250
Input Load setting R, invalid range <0.05	Set to value 0.04	Ignore input, retain previous value
	Set to value 0	Ignore input, retain previous value
	Set to value -1	Ignore input, retain previous value
Input Load setting R, invalid range >250	Set to value 250.01	Ignore input, retain previous value
Input Load setting R, invalid input: characters	Set to value character 'a'	Ignore input, retain previous value

testing the user input are shown on Table 5 and the cases used for testing the communication are shown on Table 6.

6.3.1. Final integration testing

For the final integration test the plugin was connected to the relay setting tool and used to draw various graphs to test the functionality. Because the product was now fully integrated

there were only two sources of input; input provided by the user and the data read from file. The communication between the plugin and setting tool is now completely handled internally by the program and is not visible to the outside. As the file-reading was also tested thoroughly with both the relay setting tool and the plugin and testing it was thus omitted from the final integration testing. However the available user input is different from the perspective of the full program and the individual modules and it needed to be tested again with different parameters.

The final integration testing was performed in a blackbox-manner. The equivalent classes and test-cases are listed on the Table 7. Here are however only listed the user input to the plugin and one example of the inputs to the relay setting tool: Load setting R. Similar equivalence classes and tests were designed also for the following input variables: Load Angle, Line Angle, Enable for Z1<, Z1< Direction mode, Z1< X Setting, Z1< R Setting,

Figure 34. The test-setup which was used to perform the testing and demonstrations.

Enable for Z2<, Z2< Direction mode, Z2< X Setting, Z2< R Setting, Enable for Z3<, Z3< Direction mode, Z3< X Setting, Z3< R Setting, Enable for Z4<, Z4< Direction mode, Z4< X Setting, Z4< R Setting, Enable for Z5<, Z5< Direction mode, Z5< X Setting and Z5< R Setting. Correct behaviour to the inputs were different types of graphs and they were not constructed in advance for the purposes of comparison because their validity could be easily checked in real time by the person responsible of doing the testing.

6.4. Comparison between the old and new

Finally a demonstration encompassing two tests was performed in order to show the system brings something new to the existing system, something that was not previously available and can be benefited from. Because of this, the first test was performed by using the unaltered system, to establish a baseline to which the results of the second experiment

Table 8. Values set to distance protection functions for the experiments.

Parameter	Value
Line angle	75
Load angle	10
Load resistance	4
Zone 1 enabled	true
Zone 1 direction	forward
Zone 1 resistance	1
Zone 1 reactance	1
Zone 2 enabled	true
Zone 2 direction	forward
Zone 2 resistance	2
Zone 2 reactance	2
Zone 3 enabled	true
Zone 3 direction	forward
Zone 3 resistance	3
Zone 3 reactance	3
Zone 4 enabled	true
Zone 4 direction	reverse
Zone 4 resistance	3
Zone 4 reactance	3
Zone 5 enabled	true
Zone 5 direction	undirectional
Zone 5 resistance	4
Zone 5 reactance	4

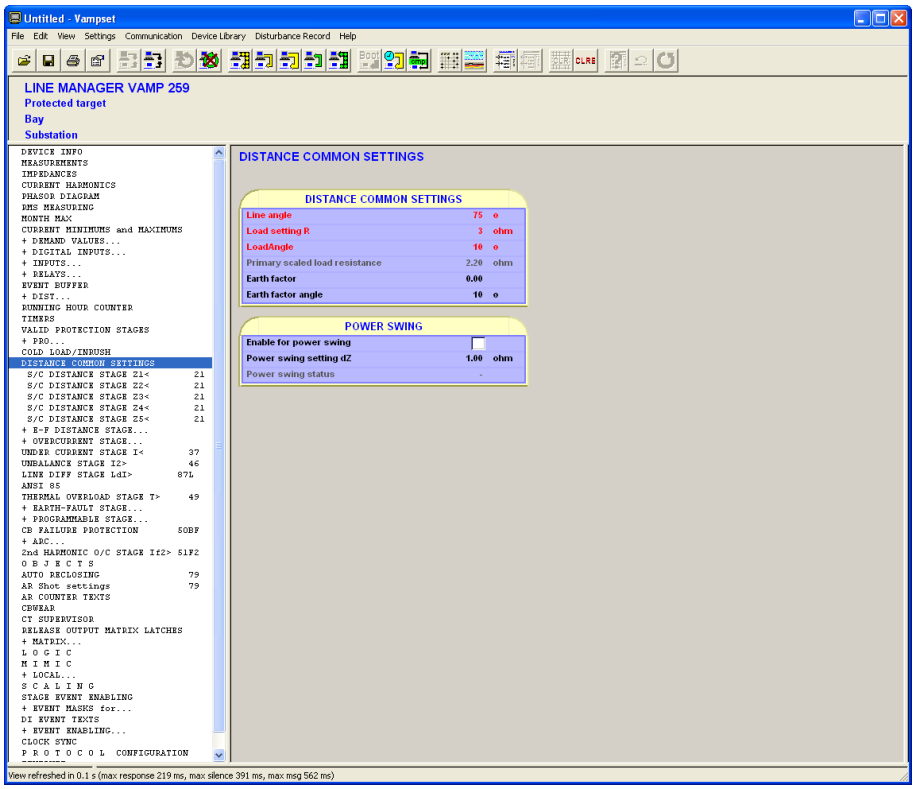

Figure 35. Results of the first experiment, part a

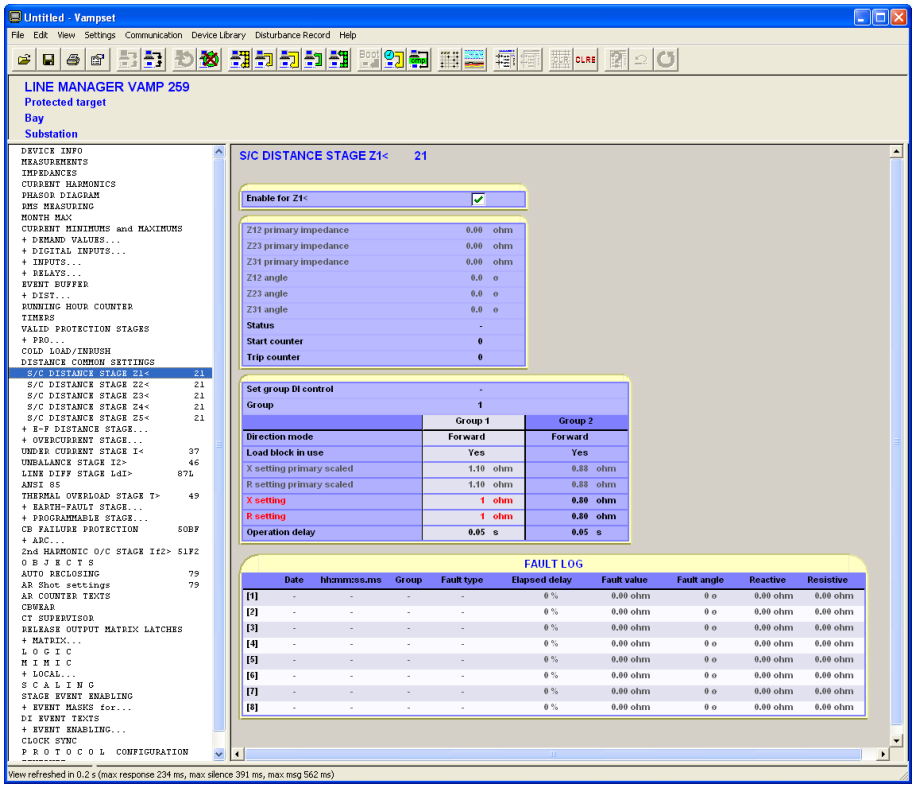

Figure 36. Results of the first experiment, part b

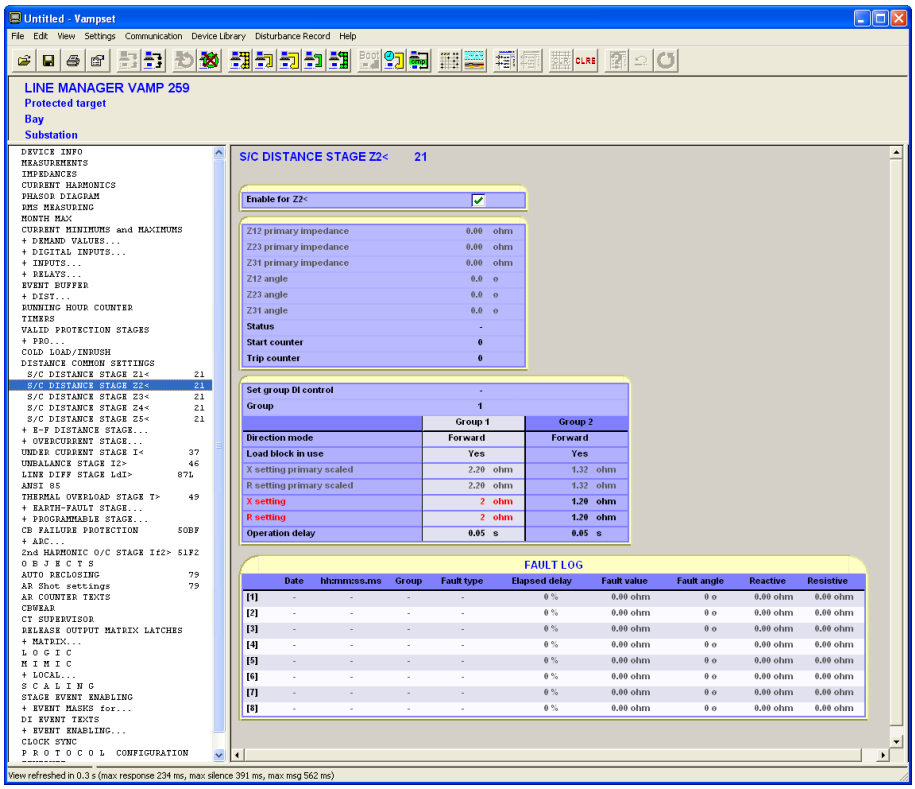

Figure 37. Results of the first experiment, part c

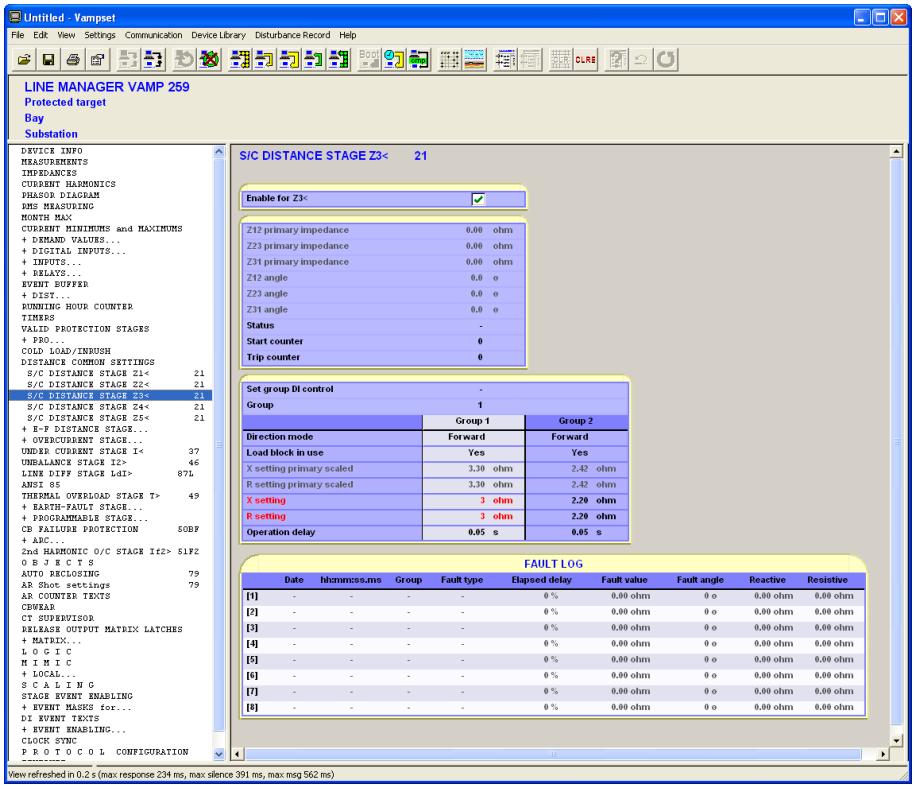

Figure 38. Results of the first experiment, part d

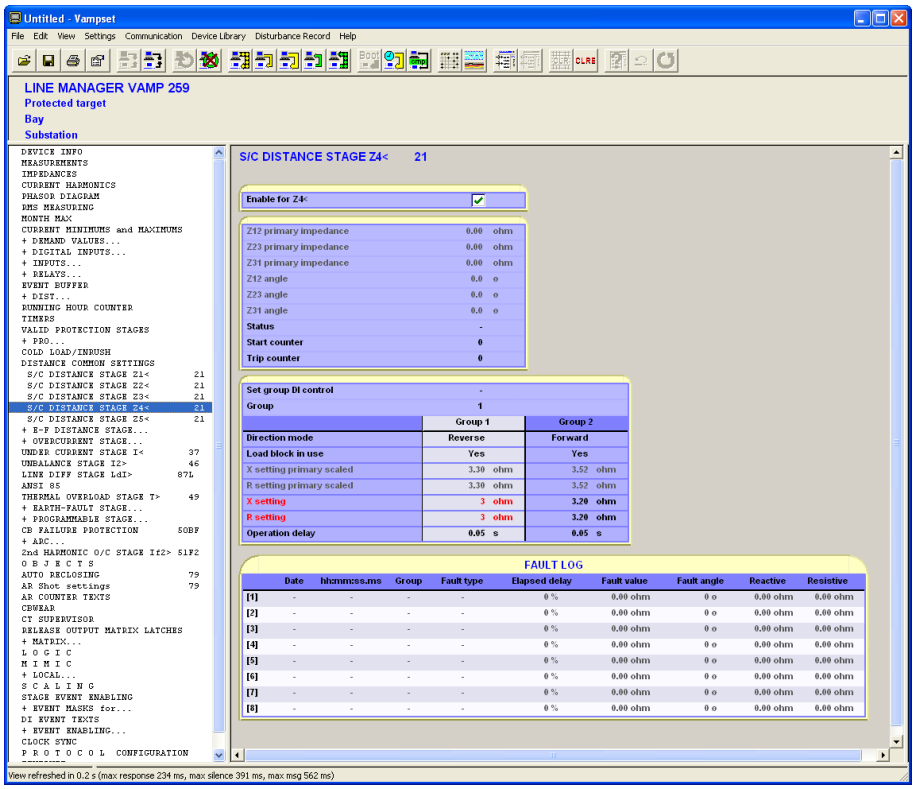

Figure 39. Results of the first experiment, part e

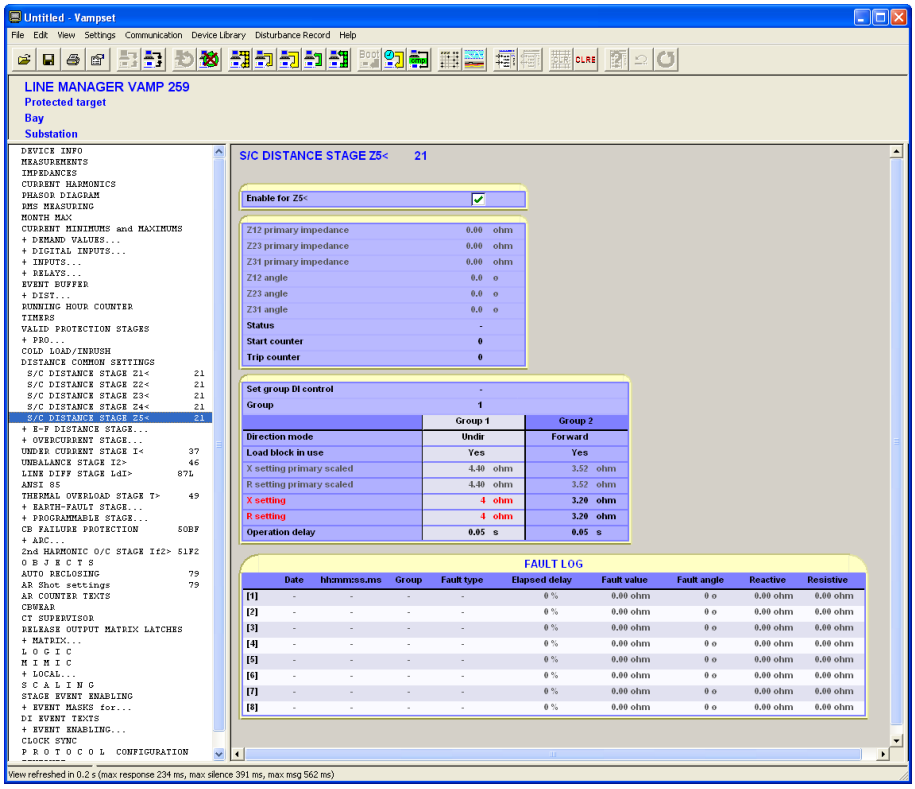

Figure 40. Results of the first experiment, part f

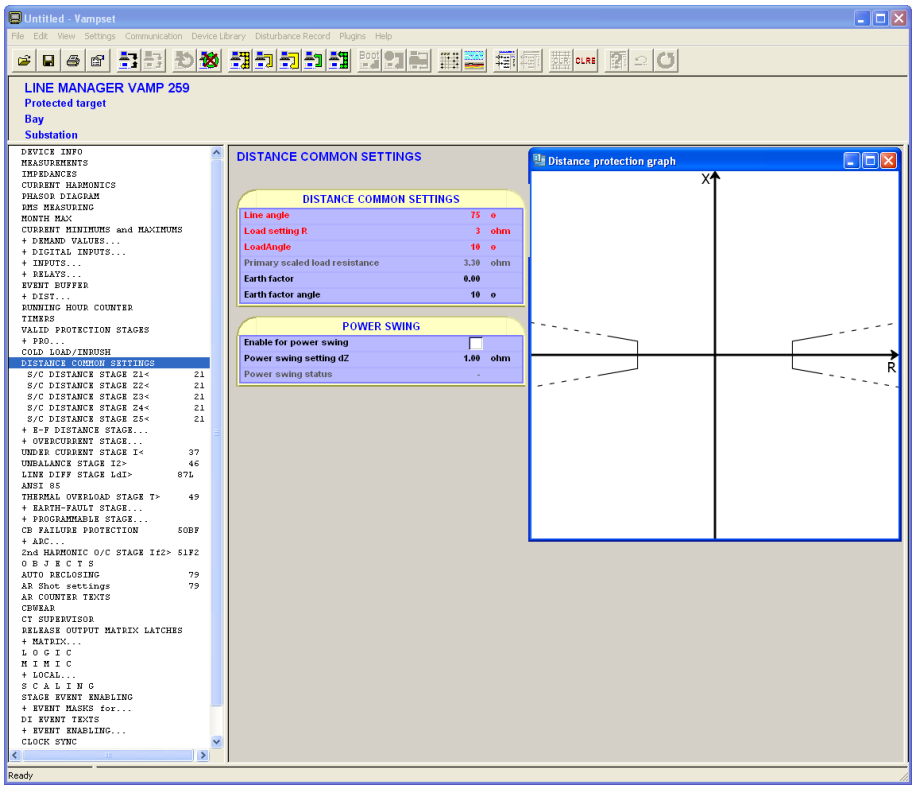

Figure 41. Results of the second experiment, part a

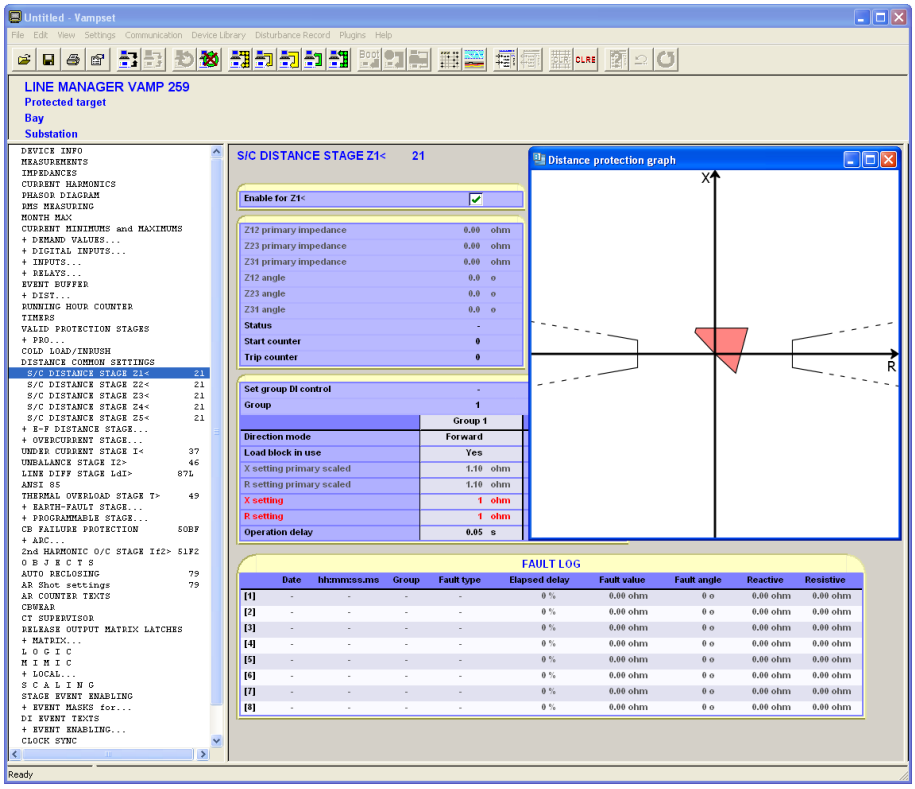

Figure 42. Results of the second experiment, part b

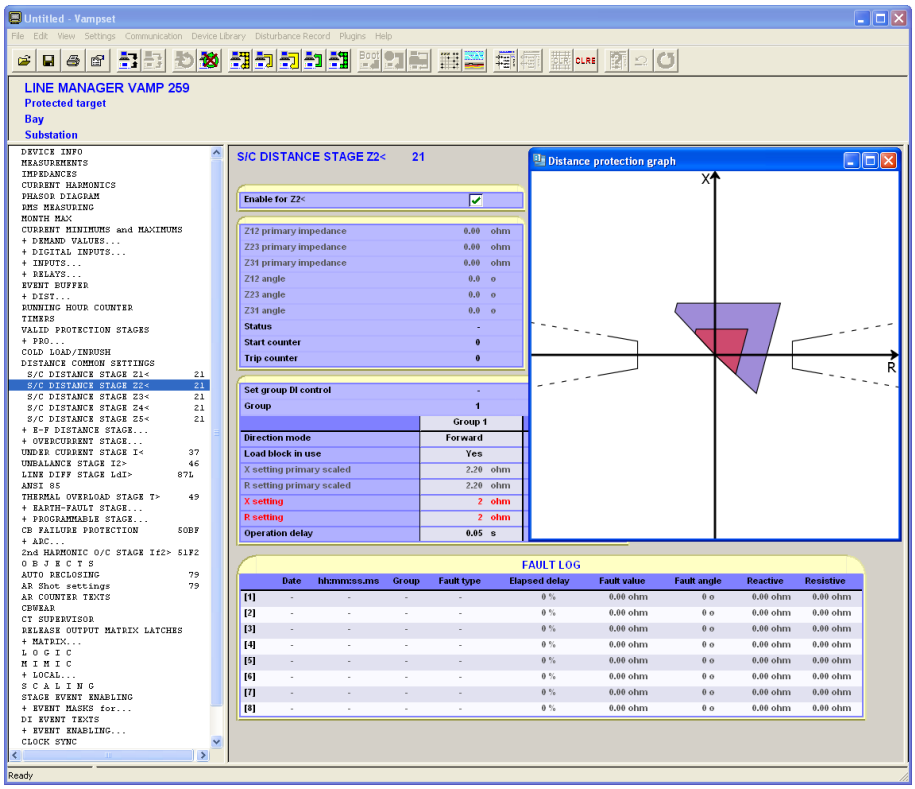

Figure 43. Results of the second experiment, part c

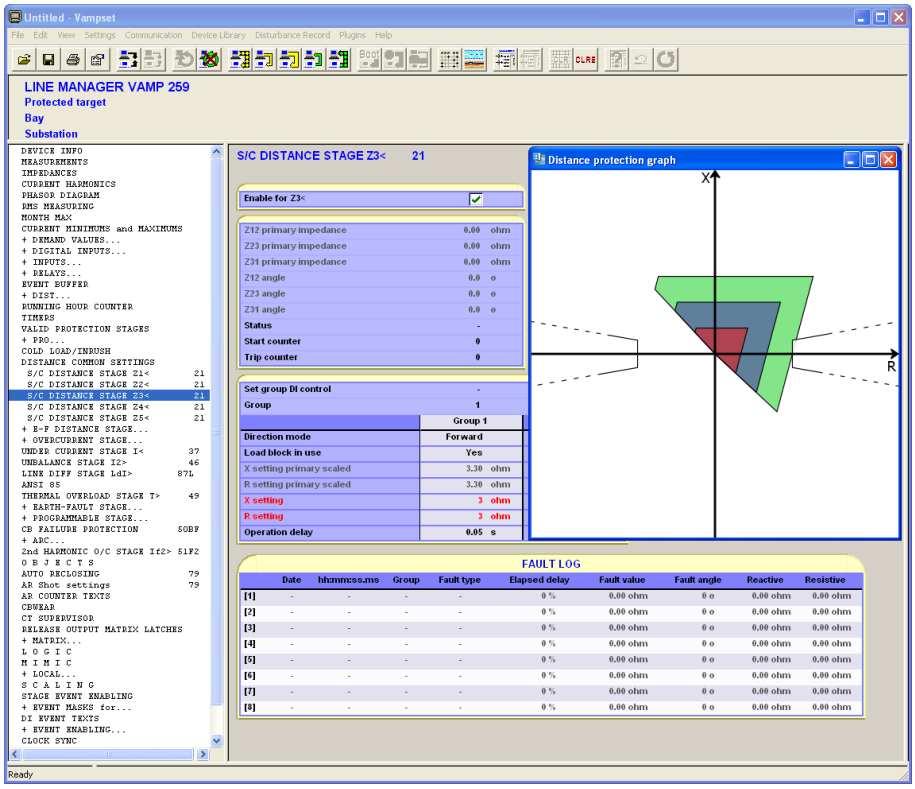

Figure 44. Results of the second experiment, part d

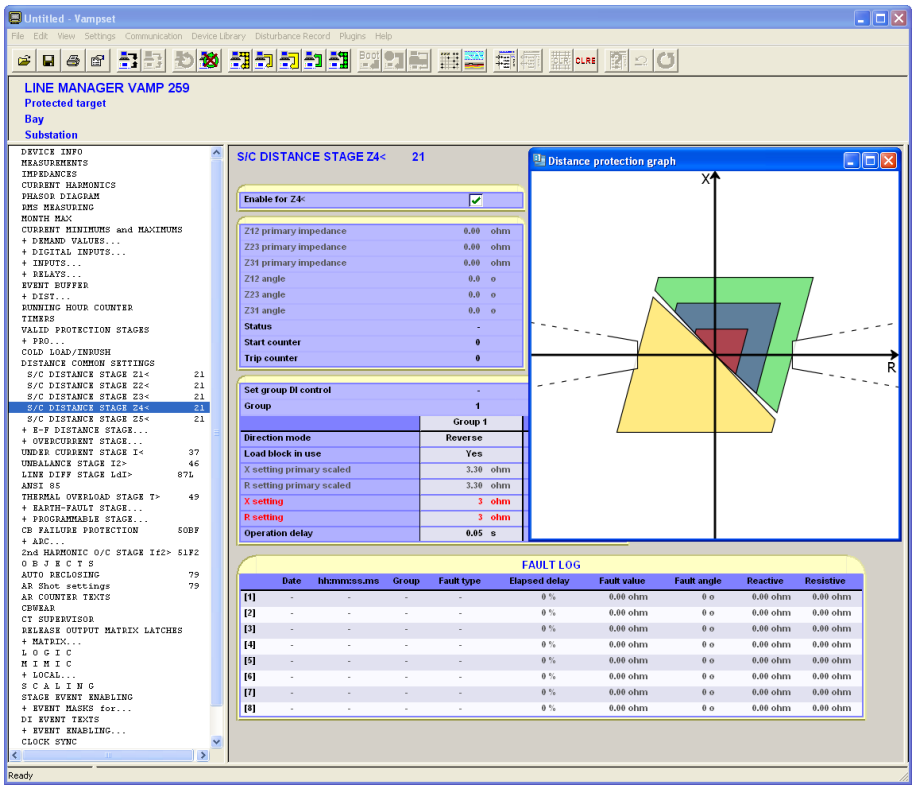

Figure 45. Results of the second experiment, part e

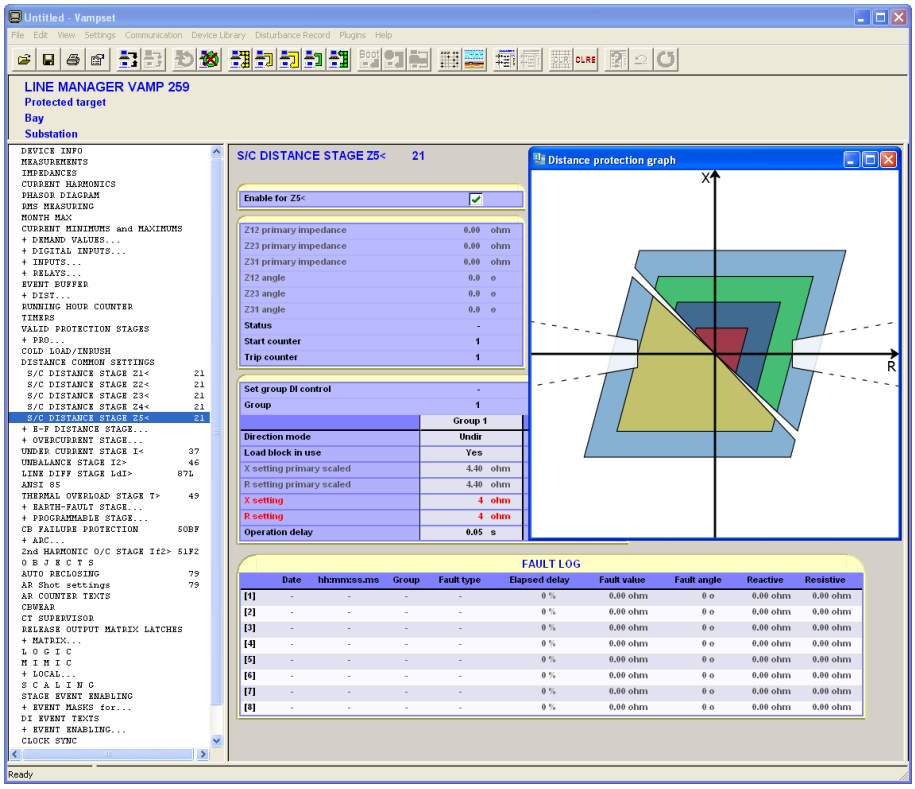

Figure 46. Results of the second experiment, part f

could be compared to. The second experiment was then done by using the new system developed in this thesis.

The experiments were performed by using a VAMP 259 protection relay, a feeder manager to be precise. In addition to the relay, an ordinary laptop was required. Lenovo Thinkpad T400 with 64-bit Windows 7 was chosen. FTDI ES-U-1001-R100 USB to RS232 adapter was used to convert the serial interface on the relay's front panel to and USB-connection. VAMP VX003-3 cable was used to make the connection between the relay and the USB converter. Obviously a power-supply was also required for both the PC and the relay. Picture of the test-setup is shown on Figure 34.

The experiments them self were a simulation of a possible real-life scenario; the process of configuring the distance protection function. The setting values for the distance protection functions were chosen arbitrarily, as the purpose of the experiments was not to test the functionality of the distance protection function, but to test the process of configuring it.

The demonstrations were conducted by first building the test setup shown on Table 8. The relay setting tool was then used to make connection to the relay, as described previously in the Section 2.4.3. Finally the relay setting tool was then used to configure the distance protection function, while observing the functionality of the program. The following values were set to the various parameters of the distance protection functions:

The results of the baseline-experiment are shown on Figures 35- 40. The experiment was to use the Vampset-tool to configure the distance protection function with the parameters described in the previous chapter. Part (a) of the Figures 35 shows the setting of the line angle, load angle and load resistance. Parts (b)-(f) on the Figures 36- 40 show the setting of the enabled-, direction-, resistance- and reactance-parameters on the distance protection zones from 1 to 5. No graph was produced on this experiment.

The second experiment was to set repeat the steps in the first experiment, but this time by using the tools developed in this thesis instead of using the old version of the relay setting tool. The results of the second experiment are shown on Figures 41- 46. Part (a) of the again shows the setting of the line angle, load angle and load resistance. Parts (b)-(f)

show the setting of the enabled-, direction-, resistance- and reactance-parameters on the distance protection zones from 1 to 5.

As expected, a graph representing the distance protection functions was produced and displayed to the user. The graph was also automatically updated after new settings were provided to the program.

7. CONCLUSIONS AND FUTURE

The purpose of this work was to constructively show that it is possible develop a system for visualizing protection relay functions. The required software was successfully developed and thus the goals were achieved. An application of the well-known waterfall software development models was used and the required system was then designed, implemented and tested. The user interface was designed by using a prototyping process where prototypes of the user interface were made, then given to experts for evaluation and new prototypes were then constructed based on the feedback. This process was repeated until all of the available experts were satisfied with the program. Software was also tested by means of normal black box and white box testing methods; equivalence partitioning and boundary value analysis techniques were used to ensure a proper coverage of the testing process. As the program was developed for a private company, the source codes could not be released to public domain, but the principles and inner workings of the developed methods and algorithms were explained in detail.

It would seem that providing visual aids for configuring the protection relays would make the process more easy and intuitive and that is the ultimate goal of the work done in this thesis. The experiments performed here however do not provide very strong evidence towards this. Services of various experts were used in order to determine that the user interface of the system is meaningful and does its purpose. A separate case-study would however be required in order to find out actually how useful the new system is compared to the old one. This kind of study was out of the scope of this thesis and it was not required by the target-company for which this work was done.

Visualization of the distance protection function was implemented and as there is a number of other protection functions, a natural extension to this would be to use the system developed here as basis and create visualizations for other protection functions as well. Work for this is in fact already ongoing on some level, but due to time constraints it could not be included in this thesis.

The systems developed here have been integrated to the target-company's protection relay setting tool and thus they will be delivered to their internal testing and finally provide the

tool to their customers when the next version of the tool is released. Feedback from the customers will probably also affect the way in which the tool is to be developed in the future.

We suggest that the target company would continue to develop the user interface of the relay setting tool and the system developed in this thesis can easily be utilized for this. The developed API is also well suited for developing other additional features which might not even need to be directly related to the user interface, but could improve the program in other ways.

REFERENCES

Blackburn, J. Lewis (1987). *Protective Relaying - Principles and Applications*, vol. 37 from series *Electrical Engineering and Electronics*. New York: Marcel Dekker, Inc. ISBN 0824774450.

Cook, Vivian (1985). *Analysis of Distance Protection*. Letchworth, Hertfordshire, England: Research Studies Press Ltd. ISBN 0471907499.

EIA (1969). *Standard RS-232-C - Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange*. Washington: Electronic Industries Association.

Future Technology Devices International Limited (2009). *Datasheet: Vinculum VNC1L Embedded USB Host Controller IC Datasheet Version 2.00* [online]. Glasgow, United Kingdom: Future Technology Devices International Ltd [cited 28.2.2010]. <URL:http://www.vinculum.com/documents/datasheets/DS_VNC1L_V201(FT_000030).pdf>.

Haikala, Ilkka & Jukka Märijärvi (2004). *Ohjelmistotuotanto*. Tenth edition. Helsinki: Talentum. ISBN 9521408502.

Hewitson, Les, Mark Brown & Ramesh Balakrishnan (2005). *Practical power system protection*. Burlington, Massachusetts, USA: Elsevier. ISBN 0750663979.

IEC (2002). *International Electrotechnical Commission Electrical and electronic terminology database - Elementary relays / Relay types, IEV 444-01-01* [online]. [cited 5.5.2010]. <URL:http://www.electropedia.org/iev/iev.nsf/display?openform\&ievref=444-01-01>.

IEEE (1990). *610.12-1990 - Standard Glossary of Software Engineering Terminology*. Institute of Electrical & Electronics Engineers. ISBN 155937067X.

Jachmann, Thomas & Andreas Feuerer (2008). *Product manual: Ordering process for DIGSI 4 Scientific* [online]. Nuernberg, Germany: Siemens Aktiengesellschaft [cited 23.6.2011]. <URL:http://www.energy.siemens.

com/co/pool/hq/automation/power-transmission-distribution/protection/siprotec/operating-programs/DIGSI4_Scientific_Covering_letter_eng.pdf>.

Kaner, Cem, James Bach & Bret Pettichord (2001). *Lessons Learned in Software Testing*. New York: John Wiley & Sons, Inc. ISBN 0471081124.

Kaner, Cem, Jack Falk & Hung Nguyen (1999). *Testing Computer Software*. Second edition. New York: John Wiley & Sons, Inc. ISBN 0471358460.

Mason, C. Russel (1956). *The Art and Science of Protective relaying*. John Wiley & Sons, Inc. ISBN 0471575526.

Paavola, Martti (1964). *Sähkölaitosten suojareleet*. Porvoo, Finland: WSOY.

Pfleeger, Shari Lawrence (2001). *Software Engineering: Theory and Practice*. Second edition. Upper Saddle River, New Jersy, USA: Pearson Prentice Hall. ISBN 0130290491.

Royce, Winston W. (1970). Managing the development of large software systems. *WESCON Technical Papers* Vol 14, A/1–1–A/1–9, Los Angeles, August 25–29, IEEE WESCON.

Schneider Electric Ltd. (2010). *2010 Annual Report* [online]. Rueil-Malmaison Cedex, France: Schneider Electric SA [cited 31.8.2011]. <URL:http://www2.schneider-electric.com/documents/presentation/en/local/2011/03/schneider_electric_annual-report-2010.pdf>.

Sells, Chris & Ian Griffiths (2005). *Programming Windows Presentation Foundation*. Sebastopol, California, USA: O'Reilly Media, Inc. ISBN 0596101139.

Siemens (2008). *Product manual: SIEMENS SIPROTEC System Description* [online]. Munich, Germany: Siemens Aktiengesellschaft [cited 23.6.2011]. <URL:http://siemens.siprotec.de/download_neu/devices/1_General/System_Manual/SYSTEM_MANUAL_B3_SIPROTEC4_DIGSI4_EN.pdf>.

Tanenbaum, Andrew S. & Maarten Van Steen (2007). *Distributed Systems: Principles and Paradigms*. Second edition. Upper Saddle River, New Jersy, USA: Pearson Prentice Hall. ISBN 0132392275.

Vainionpää, Jani (2011). *VAMPSet Update v2.0*. Vamp Ltd. internal documentation, unpublished, stored at Vamp Ltd. head office in Vaasa, Finland.

Vamp Ltd. (2009a). *Customer journal no. 2/2009* [online]. Vaasa, Finland: Vamp Ltd. [cited 5.5.2010]. <URL:http://www.vamp.fi/Customer%20journals/Protection%20and%20control%202_2009.pdf>.

Vamp Ltd. (2009b). *Lehdistötiedote: Maailman johtava valokaarisuojaustekniikan yritys Vamp yrityskaupalla AREVA T&D:n omistukseen* [online]. Vaasa, Finland: Vamp Ltd. [cited 12.5.2010]. <URL:http://www.vamp.fi/Press%20release/Press%20release%20Areva%20VAMP%20in%20Finnish.pdf>.

Vamp Ltd. (2011). *VAMP 259 Line Manager - Operation and configuration instructions, Technical description* [online]. Vaasa, Finland: Vamp Ltd. [cited 1.9.2011]. <URL:http://www.vamp.fi/Manuals/English/VM259.EN007.pdf>.

Virtala, Tero & Jarkko Holmlund (2009). *VAMPSet Update*. Vamp Ltd. internal documentation, unpublished, stored at Vamp Ltd. head office in Vaasa, Finland.

Watson, Karli, Christian Nagel, Jacob Hammer Pedersen, Jon Reid, Morgan Skinner & Eric White (2008). *Beginning Microsoft Visual C# 2008*. Indianapolis, Indiana, USA: Wiley Publishing, Inc. ISBN 9780470191354.