Designing and Implementing Communication and a Graphical Interface for Antenna Test Devices

Daniel Stålnacke

April 16, 2010
Master’s Thesis in Computing Science, 30 ECTS
Supervisor at CS-UmU: Jonny Pettersson
Examiner: Frank Drewes

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

Esrage Space Center, located in Kiruna, is a company which specializes in the area of satellite service as well as rocket and balloon launches. To communicate with the satellites Esrange use antennas that is between eight and thirteen meters in diameter. The antennas must always be ready to operate which is why there is a need for an application to test the antennas when they are not used.

This master’s thesis work aims to develop such a system that tests the antennas when they are not communicating with satellites. This report contains background information about the problem and system requirements as well as the solution and description of the developed system. It also contains an in-depth study about similar applications that has been developed by others.

To solve the problem, a client-server application was developed. The server controls the test devises used to test the antennas and the client gives the users a GUI to control the server and its functions. An automated system called OASIS, developed at Esrange, was included in the solution to make it possible to automate some of the core tests.
# Contents

1 Introduction  

2 Problem Description  
   2.1 Problem Statement ............................................. 3  
   2.1.1 OASIS .................................................. 4  
   2.2 Purposes .................................................. 5  
   2.3 Goals ................................................... 5  
   2.4 Limitations ............................................... 6  

3 Distributed Test Control Systems  
   3.1 Distributed System ............................................. 7  
      3.1.1 Different Kinds of Distributed Systems ................. 8  
      3.1.2 Disadvantages and Difficulties in Distributed Systems .... 10  
      3.1.3 How to Address the Difficulties .................. 14  
   3.2 Distributed Systems Interacting with Test Devices ............ 18  

4 Method  
   4.1 Preliminaries ............................................... 21  
   4.2 How the work was done ..................................... 21  

5 Results  
   5.1 Test Device ............................................... 25  
   5.2 The Developed System ....................................... 27  
      5.2.1 Server .............................................. 27  
      5.2.2 Client ............................................. 33  
      5.2.3 Discussion ...................................... 35  

6 Conclusions  
   6.1 Limitations ............................................... 37  
   6.2 Future Work ............................................... 38  

7 Acknowledgements  

iii
References \hfill 41
A System Requirements \hfill 43
B Planning \hfill 45
# List of Figures

2.1 An overview of how the old system is structured. ........................................ 4
2.2 An overview of the connections with the different parts in the planned system. ........................................ 6

3.1 An example of events over time in a distributed system. ......................... 12
3.2 An example of logical clocks used in a distributed system. ..................... 13
3.3 An example of vector clocks used in a distributed system. ..................... 13
3.4 A design where authentication would be possible. ................................. 16
3.5 Two clients communicating with a sequencer before sending messages to server. ........................................ 18

4.1 How the structure of the server was planned to look like. ..................... 22

5.1 An overview of the test devices. ............................................................... 26
5.2 The signal generator web interface. ......................................................... 26
5.3 Basic structure of one server connected to several clients. ..................... 27
5.4 The classes in the server. ............................................................ 28
5.5 The first tab in the client GUI. ............................................................... 33
5.6 The second tab in the client GUI. .......................................................... 34
5.7 The third tab in the client GUI. .............................................................. 35
LIST OF FIGURES
Chapter 1

Introduction

Esrange Space Center, from now on referred to as Esrange, is a part of the Swedish Space Corporation (SSC). Esrange is located 45 km from Kiruna. Their main activity is satellite service, rocket and balloon launches, flight test services and ground based scientific instrumentation. The satellite operational division has several big antennas, eight to thirteen meters in diameter, that is used to communicate with satellites passing over. The satellites sends and receives information from the antennas and it is important that the antennas always are ready to operate. Esrange has a set of test devices that sends test signals to the antennas to make sure they operate as intended. The software used to control the test devices is outdated and not reliable, and Esrange is requesting a new software solution to control the device.

This report describes a client-server application as a solution to control the antenna test devices. A server in this case is an application that controls the test devices and provide clients with services to read information. The client gives the Esrange operators a graphical interface to operate with the system, from where they can send commands and watch the status of the test devices. An automated system called OASIS is described in the report, and is used to command a server located at the Esrange Satellite Station, ESS, to take action in different ways at specific times. The client-server application described in this report interacts with OASIS to make automatic test at a regular basic. This makes it possible to run test without an operator supervising it.

The report is divided into seven chapters. In chapter two the problem is described as well as the background information and planned solution to the problem. Chapter three focus on other similar projects done earlier. Those projects are summarized as an in-depth study to provide ideas and good designing decisions for this project. Chapter four includes descriptions about how the work was done and the methods used. Chapter five describes the results including descriptions of the system developed. Conclutions drawn from the project are presented in chapter six.
Chapter 2

Problem Description

The satellite service team at Esrange uses several antennas located within the Esrange area, and these are used to communicate with satellites passing over. A satellite passes over between once a day up to several times a day. When this occurs the satellite station both sends and receives information from the satellites. It is crucial to have the antennas assigned for the communication functioning when there is a limit of time in which the communication can take place.

To make sure the antennas are working as intended ESS have a set of test devices on a mountain top called Skaitevaara which is located approximately 5.5 km from the satellite station. The test devices sends signal similar to those satellites sends which will simulate a connection between a specific antenna and a satellite. The test devices can send signals on S-band and X-band which is, respectively, two to four GHz and eight to twelve GHz. These are the band in which the satellites communicate with the antennas. The test devices sends signals to the antennas to simulate a connection and an operator can then check the software developed for the antennas to see if the communication was correct. Testing is used to avoid missing information exchange with a satellite by, when error occurs, change to a different antenna for communication. ESS provide satellite services to other companies and it is crucial that all communication with the customer’s satellites is competed correctly.

2.1 Problem Statement

Already existing software that controls the test devices is used by ESS operators before this project started. The software is called Skaite Control and it allows the operators to communicate with the test devices using radio modem. The major problem with today’s system is that the radio modems has communication issues when it is bad weather or when the radio modem’s antennas are covered in ice. Esranges Science Service Division has a wireless network that can be used as a communication channel. This means that the new software does not have to use radio modems for communication. The software used today is also lacking the possibility to set all the possible parameters in the test device. ESS has an automated system called OASIS and they want to integrate it with the test device software which is not a feature of today’s system. The system used today is not a client-server architecture, Skaite Control connects directly to the radio modem which means that only one user can be connected to the test device. This user has to disconnect in order for another user to connect. Figure 2.1 shows an overview
of the old system. A new device is planned to be put up at skaitevaara and that is a power switch which will enable the operator to turn other equipment on and off from the application. This means that the operators can reboot a single device directly from the client application if necessary.

![Diagram](image)

Figure 2.1: An overview of how the old system is structured.

**PC** is a computer located at the satellite station with the *Skaite Control* application installed and it is used by the operators to communicate with the test device through the radio modem.

**Radio modem** is used to send signals between the satellite station and Skaitevaara where the test device is located.

**COSII** stands for Code Operated Switch II and it is used to switch from a DTE (Data Terminal Equipment) and a DCE (Data Circuit-terminating Equipment). It takes signals from the radio modem and converts it to something the test device can understand and vice versa.

**CTT** is a test transmitter that distributes a VGT X-band signal through the antenna switch to one of the antennas. This is one of the two devices that create signals that are sent to antennas for testing, the other one is the Anritsu device.

**GPIB-23CT-A** is a converter that converts digital signals into IEEE-488 standard, also called General Purpose Interface Bus (GPIB), which Anritsu understands.

**ANRITSU** is a radio frequency signal generator that transmits a S-band signal with a frequency band from 10 kHz to 2700 MHz. The signal can however be up converted to 7000 MHz to 9700 MHz using an up converter in the ESAS.

**ESAS** is an antenna switch. It switches the output signal between small antennas pointing at different satellite antennas.

### 2.1.1 OASIS

Esrange has an own developed system called OASIS which automates the monitoring and control of other systems within the Esrange satellite station. OASIS has a schema for every server it is monitoring, the schemas contains information about what a certain
server must do at a specific time. The schema is sent to the servers and they contain macros and timestamps, a macro is a predefined word which the applications translate into one or several actions that has to be executed. The timestamp is used to tell the servers to when, in time, the macros has to be executed. OASIS can also send a macro as a single command to a server, this macro has to be executed instantly. A server connected to OASIS receives a special macro together with a file name to receive a new schema file. The server then uses FTP, File Transfer Protocol, to retrieve the new file from OASIS. Each server must send status messages every second to assure OASIS that it is running and working as intended. All communication with OASIS is done using UDP, User Datagram Protocol, except for file transferring that uses FTP.

2.2 Purposes

The purpose of this master’s thesis work is to improve an already existing system, by building a new software to it. Today’s system is both old and unreliable which creates problems for the operators when using it. The system does not support OASIS or any other automated tests which means that the operators must do the tests manually, this is both time consuming and unnecessary. All parameters available to change in the test device cannot be changed in the old software, just a handful of the most important ones.

The purpose is to address all problems in the old systems and develop a new software with the functions of the old system and solutions to the problems related to that system.

2.3 Goals

The goal of this project is to develop a new software which communicates with the test device using Ethernet to avoid weather related problems and develop an improved graphical user interface. The new software aims to support changing of all the parameters in the test device and support interaction with OASIS. Allowing OASIS to send commands to the system will make the testing of the antennas more automated, which makes it possible to run tests during the night when there is fewer satellites pass than during the day. The planned software will be built on a client-server architecture which was decided after conducting an in-depth study presented in chapter three, this means that there is a server that is always running and that the users of the systems connect to the server through a client. The goal is to let he server communicate with the test devices to allow communication on all times from any computer within the ESS. Today’s system uses a local application that directly connects to the test device. The client must have a GUI, Graphical User Interface, to make it easy for the operators to connect to the server from their local computers. When an operator is connected to the server via the client application and runs it local, OASIS can not be able to execute any macros on the server. The client must have two state options where it can either be executed locally which means that it can send message to the test device, or remote where the operators only can check the status of already performed commands or antenna status. The goal is be able to communicate with all devices on skaitevaara using TCP/IP.

The server application will be implemented in the program language C++ and the client in TCL/TK. The complete set of requirements can be found in the appendix A.

The figure 2.2, is an overview of the planned system.

GUI is the client application used by the operators. This application communicates only with the server through TCP/IP.
OASIS is ESS own developed system. OASIS sends and receives messages from the server through UDP and distributes files through FTP. OASIS communicates only with the server.

Server is the application controlling everything. It receives messages from OASIS and the client and sends status messages back, the server sends commands to the test devices using TCP/IP. The messages from the server to the test devices will go through the main building at Esrange.

MB is the main building on Esrange. The communication between the satellite station and the test device will go through this location. The main building will communicate with the test devices using a wireless network.

Test devices is the test devices located on the top of skaitevaara. All the components in the test device will be explained in the result part of the report.

2.4 Limitations

The major limitation in this project is time, the project should only last for 20 weeks in total which includes problem introduction, planning, implementation, testing, writing this report as well as making the system go live for the operators to use. There are not many limitations in the planning phase but as the project is scaled with the time given limitations can arise. System limitations are discussed in the conclusions, chapter six.
Chapter 3

Distributed Test Control Systems

The software *Skaite Control*, developed by ESS earlier, has several design issues. Besides the fact that the radio modem used to communicate have some weather related connection problems, the software itself needs some re-design. The largest design issue in *Skaite Control* is that it is built as a local application which means that the graphical user interface and the underlying test device communication is in the same application. The application connects to the test device directly through the radio modem which prevents other instances of the same software to connecting to the test device. This is an issue if several users need to check the status of the test device at the same time, or if one user needs to change any state in the device while another user is monitoring it.

With the local system structure comes other problems as well. The software together with the GUI have to be constantly running on a local computer to have a constant connection to the test device, the result of this is that a special computer located at the ESS must be used at all time to provide such connection. A second problem with a local system structure is that if automated tests would be implemented in the system, which the current system does not support, the software has to be running to be able to execute the tests. The issue of using a local system structure can be solved if the underlying communication with the test device is performed by one node and the graphical user interface is displayed by another. These types of systems are called distributed systems.

3.1 Distributed System

Coulouris et al. (2005) defines a distributed system as "one in which components located at networked computers communicate and coordinate their actions only by passing messages". This means that all the components in a system are executing their part at physically different locations which are connected through a network and communicates using messages. The parts of the system does not have to be located at physically different location and can be executing on computer located in the same room or even within the same computer, but they will still communicate using messages over a network. The benefits of using a distributed system, according to Alegría and Ramos (1997), is that regardless of the physical location of the resource and the user will data, programs and equipment be available for anyone on the network. This is a desirable feature for the
new skaitevaara test device software because it will allow several users to integrate and monitor the software simultaneously which in fact is one of the requirements given by ESS. This means that a single computer located at the operators office does not have to constantly run the software to monitoring the system, if a task has to be done or an error occur an operator can start the client side of the system and perform a task and then close it again. The underlying software will then be located on a server which will be constantly connected to the test devices, this means that if any user wants to use, change or check the status of the system they just start the client side which connects to the server. This means that the user can get an instant access of the system without waiting for the system to connect to the test device.

3.1.1 Different Kinds of Distributed Systems

There are several different structural designs of distributed systems. In this section several structural designs will be discussed where some are new and some are old and well known to figure out which design fulfills the requirements for this project.

Token Ring

Token ring networks are one of the oldest system structures in distributed system and were created to increase the performance and network speed in distributed systems. A token ring network is, according to Lim et al. (1991), in theory a network located at a LAN consisting of n nodes connected point-to-point to each other forming a circle. A token is sent around among the nodes in the ring to determine who is allowed to send messages in the network. At this point conclusions can be made that a token ring network is not suitable for this project as there is no need to have the clients connected as a circle to improve the performance in the network. Networks of today have no problem allowing several clients sending messages simultaneous without slowing the network speed down.

Peer-To-Peer

According to Milojivic et al. (2002) peer-to-peer, also called P2P, is a class of systems that performs functions in a decentralized manner on distributed resources. The resources can be computing power, network bandwidth or data storage. The functions can be data or content sharing, distributed computing, platform service, or communication and collaboration. P2P systems became popular after a program called Napster was distributed through the web in the early 2000 which made it possible for users to share files through the Internet.

Peer-to-peer systems have the advantage that it is decentralized which means that each process communicates directly to the receiver and not using a server which in a big system can become a bottleneck. In P2P system each node works as both a client and a server that distributes resources, which in this projects system would mean that each client must be able to communicate with the test device, change its parameters and provide other clients with information about the test device. No further analysis must be made to conclude that a peer-to-peer system is not suitable for this project.
3.1. Distributed System

Client - Server

According to Reese (2000) is client-server often used as a general expression for an architecture where processing are divided among two or more processes often located at physically different machines. His example is a database system where the server is storing the data and the client gets and creates data. There are two different types of client-server architectures; two tier and three tier, often called n tier.

Two tier client-server architectures are the most commonly used and consist of multiple nodes with the same user interface which communicates with a centralized process that provides some sort of service to the nodes. The multiple nodes are clients and the centralized process is the server. A three tier client-server architecture is used to solve more complex systems where the third part is used between the client and the server. One example is a middleware which service data requests between a user and a database located at multiple servers. There can be cases of several nodes between the client and the server which are then called layers and the architecture a n tier network.

A client-server architecture is a good choice for design as it will allow the user interface to be separated from the underlying test device communication which was one of the problem with Skaitel Control discussed in chapter two. The client side would provide a user with a graphical user interface to display system status as well as options to change the status in the system by sending messages to the server. The server then does not have to provide users with a graphical user interface but rather focus on the communication with the test device, OASIS and the clients. The server can be located at any place within the Esrange Satellite Station and still provide the operators with a service to operate within the system. As the system complexity for this project is low does it not require the architecture to be a three tier system. So far this is the best choice for a system architectural design in this project.

Web Services

Web services are a part of the field of grid computing and service-oriented architecture and can be viewed as a subsection of client-server architectures. Gannon et al. (2002) explains that web services combines for example computers, networks, on-line instruments, data servers or sensors together as services which is provided to the users as a collection through one single service. A common use for these applications is to collect several sub-applications in to one service which provides all services each sub-application provide. The biggest advantage about using web services in this project, mentioned by Papazoglou (2007), is the loose-coupling between the server and the users. Web services provide a interface for the user to interact with which make it possible for several different clients to interact at the same time as long as they use the same interface. This can allow ESS to change the graphical user interface if it gets outdated or needs to be re-designed.

A web interface can be built to communicate with the web service which allows users to interact with the test device using their web browser and not a specific software. There are benefits of using a web service in this project compared to a ordinary client-server structure, however the time it takes to design such a system can be overwhelming for this project. A design choice will be made further on into the project when more details have been worked out.
3.1.2 Disadvantages and Difficulties in Distributed Systems

There are not only benefits of using a distributed system and like all system structure designs follows some disadvantages. Distributed systems have many advantages as well as some disadvantages and many of the problems evolved from the disadvantages can be solved by implementing features to prevent those things from happening.

Security

Security is an issue in most systems communicating over the Internet or connected within a network which is part of the Internet, because this can allow non authorized users to access the system or sniffing the messages sent between components in the system. A system that communicates securely satisfies both of this properties according to Gasser et al. (1989): (1) authentication, which means that the receiver knows who has created a message received and (2) confidentiality, which means that the sender knows who can read the message sent.

Woo and Lam (1992) divides authentication in two parts, identification and verification. Identification is when a part of the system claims a certain identity and verification is when the identification is confirmed. There are three main types of authentication in a distributed system:

1. Message content authentication, which verifies that the message received is the same as when it was sent.
2. Message origin authentication, which verifies that the claimed sender of a message actually is the sender.
3. General identity authentication, which verifies that a claimed user is that user.

To verify message content authentication both Gasser et al. (1989) and Woo and Lam (1992) proposed that a message authentication code, MAC, is attached to the messages sent in the system. The thing with MAC is when a code has been attached to a message the code is used and cannot be used again, this means that if someone else tries to send a message with the same MAC it will be noticed. A message with a MAC cannot be changed by someone else without getting noticed, because the message is not plain text. The MAC and the message are merged together using an algorithm to assure that the message stays the same through the network. The MAC is known both by the sender and receiver which allows merging and demerging and when or if someone else get a hold of the MAC it has already been used and is worthless. This solutions allows messages to be transported through a network without being changed or sent by someone who claims to be someone else.

Authentication case two is a sub case of case three. In an environment where both communication and hosts need verification a mutual verification must take place, which is one where both part of the system communicating must verify each other’s identity. For mutual verification authentication protocols is used which make sure that both hosts authenticates themselves.

When one source is trusted, for example a server, there is only need for an one way authentication and the most commonly used is a username together with a password. This type of authentication can only be used when the other host is trustworthy.

Gasser et al. (1989) states that to assure confidentiality there has to be a secure channel, and they define a channel as “A channel is a path by which two or more entities
communicate". A secure channel is a channel for communication where confidentiality can be assured and that can be designed in two ways. The first design is to have a secure path which can for example be a physical wire that is not connected to the Internet and the second is an encrypted logical path. Identifiers are used in secure channels by the sender and the receiver for identification. A secure physical channel uses a hardware address for identification such as an I/O port number on a computer or a disk drive and block number. The encrypted secure channel uses an encryption key for identification. A message is encrypted with a key which will be the sender’s identification and the receiver decrypts the message. This type of communication can be made over an insecure physical channel because the encryption will assure confidentiality.

### Scalability

One thing that needs to be concerned about in distributed system is scalability which can become an issue within large systems. A system is called scalable if it will remain effective when a large number of resources and users are added to the system according to Coulouris et al. (2005). Bondi (2000) state that the cost of poor scalability can be quantified in, but not limited to, response time, processing overhead, space, memory or even money. The cost of this can result in harms of service and delays at the clients which will in the end result in a re-design or replacement of the system.

Bondi (2000) divides scalability in four different types as:

1. Load scalability.
2. Space scalability.
3. Space-time scalability.
4. Structural scalability.

**Load scalability** refers to a system which remains efficient during heavy loads and makes good use of available resources. There are factors that can undermine load scalability such as the scheduling of shared resources, self-expansion which means that adding more resources to the system increases its own usage as well as weak usage of parallelism. One example of weak usage of parallelism is when a system has two servers which provide the same service and one has delays because of too many request while the other has no requests. This factors needs to be avoided when designing a distributed systems to make it efficient during all times.

A system which has **space scalability** does not require unmanageable amount of memory when the number of items it supports increases. There is always a limit of how many items that can be stored in a system, but the growth of memory requirements in a system should not have a large incensement for each item. A system is said to have space scalability if it at most require memory linearly to the number of items. This type of scalability can be solved using matrix methods or compression.

**Space-time scalability** refer to a system which continues to function effective as the number of items it supports increases. To maintain search speed in the system several different designs can be used, such as hash tables or balanced trees. Design based on linear searches is not space-time scalable because linear search is slow in a large database of items.

**Structural scalability** implies that a system does not limit the growth of items in the system. If items have a specific number the number must be large enough to not limit
the number of items. One example is phone numbers where there is a finite number of possible numbers, the length of the number gives the amount of numbers able to be created. The number of digits in the phone number must be large enough to provide every user with an unique one.

Scalability is an issue in distributed system if it is not carefully designed, this must be used when designing the application for this master’s thesis work.

**Time**

Lamport (1978) discuss in his article that distributed systems cannot rely on clocks in the same way as most other parts of the world does. As parts of a distributed system communicate using messages the order of events occurring in the system can only be determined by those messages. To determined which event happened first by attaching a timestamp on a message is not the best choice because the internal clocks at different parts of the system are not perfectly synchronized which will make errors in the ordering of events.

In a distributed system where several processes are executing simultaneous and an event \( a \) occurs before an event \( b \), is called \( a \) "happens before" \( b \) which is denoted by \( \rightarrow \) as \( a \rightarrow b \). Three rules can be drawn from the happen before relationship in distributed systems.

1. If \( a \) and \( b \) are events in the same process and \( a \) comes before \( b \), then \( a \rightarrow b \).
2. If event \( a \) is a sending event and event \( b \) is the receiver, then \( a \rightarrow b \).
3. If \( a \rightarrow b \) and \( b \rightarrow c \), then \( a \rightarrow c \).

![Figure 3.1: An example of events over time in a distributed system.](image)

Figure 3.1 shows an example of how events can occur in a distributed system over time, where the horizontal line is the time line. The figure shows events when parts of the system send messages to each other where the dots represent events and the arrows denote messages. The problem here is how to determine if \( A1 \rightarrow C1 \).

There exist several solutions to handle event ordering in a distributed system. Logical clocks invented by Lamport (1978) gives the system a numerical counter that is increased for each event in the system. Each process in a distributed system keeps its own logical clock which is added to all messages sent in the system, when a process receives a message with a higher timestamp then its’ own, it changes the timestamp to the one in the message plus one. Figure 3.2 shows an example of how logical clocks are used in a distributed system, when process B receives a message its logical clock is set to process A:s logical clock plus one. When process C receives the message from process B, its’ logical clock is set to two which is less than four, it sets its clock to five.
3.1. Distributed System

Figure 3.2: An example of logical clocks used in a distributed system.

The problem with the design proposed by Lamport is that the system cannot determine if $A1 \rightarrow C1$ in figure 3.1. There are ways to solve the problems with Lamport's logical clocks which Fidge (1991) discussed in his article. Instead of logical clock he proposed a usage of vector clock which contains, for $N$ number of processes, $N$ number of logical clocks. Each clock in the vector represents one process and each process has its own vector clock. Figure 3.3 shows the same example as in figure 3.2, but with the usage of vector clocks. In systems where the components of a system is located physically close to each other, for example on the same LAN, or when time is not a big issue physical clocks may be used. This can be done if the system decides that one component has the accurate time which then is distributed to the other parts of the system. There are several different solutions to how synchronization can be realized with physical clocks described by Coulouris et al. (2005). If the system is structured as a tree and the root is the one with the accurate time for the system time can be distributed though the whole tree where each node sends the time to the nodes further in the tree. Synchronization algorithms like Christian’s method and Berkeley’s algorithm can also be used to distribute time in a distributed system.

Time is an important part in many system and the synchronization should be chosen carefully when designing the system. For system where ‘happen before ordering’ is not really a big issue is Lamport’s logical clock a good choice though it is not so complicated to implement, if time however is important should more complex solutions be used. To solve the problem displayed in figure 3.1 message ordering must be added into the system design.

**Message Ordering**

Various types of network delays can result in reordering of messages between components in a distributed system which can results in faults in the system. It can for example be
messages to change status on a server were reordered messages can make some status unchanged while the same messages in the correct order would change the status in the system. This type of errors is unacceptable in some distributed systems and designs to prevent these things from happen must be implemented. There are solutions to prevent messages from being reordered, some are easy designed while others are more complex. Coulouris et al. (2005) presents the ordering techniques described below.

**Non-ordered** is the case where no ordering mechanism is added to the system and each process executes the messages in the order that they where received. This is the design where messages can be reordered which can result in system errors.

**FIFO** stands for First In First Out which means that the first message sent by a sender must be the first executed by the receiver. If a correct process sends $m$ and then $m'$, every correct receiver that executes $m'$ must first have executed $m$. This can be done by adding a message number to every message sent which is increased by one for each message. FIFO is in other words implemented using Lamport’s logical clocks. FIFO does not means that each process execute messages from different processes in the same order. An example is if process A sends messages $A1, A2, A3$ and process B sends $B1, B2, B3$ then one process can execute the messages as $A1, A2, B1, A3, B2, B3$ and another as $B1, B2, B3, A1, A2, A3$.

**Total ordering** means that messages from each process in the system is executed in the same order in every receiving process, in other words; if a correct process executes message $m$ before $m'$, then any other correct process that executes $m'$ must first have executed $m$. Total ordering is FIFO but for the whole system and one way to accomplish total ordering is to add a sequencer to the system. Each process ready to send a message communicates with the sequencer which gives the process a message number, this message number functions as the ones in the FIFO solution.

**Causal ordering** states that if one process multicast a message $m \rightarrow m'$, then any correct process that executes $m'$ executes $m$ before. To exemplify we can say that one process D multicasting a message has executed three messages from process A, two from process B and six from process C, then all receiving processes must have executed the same amount of message or more to be allowed to execute the received message from process D. Causal ordering can be solved by letting each process have their own vector where they count how many messages they have executed from each process and every number in the vector represent one process. This is an implemented using vector clocks described in the last chapter. When one process multicast a message, it adds the vector to the message which is used by the receiving vectors to know when to execute the message.

Coulouris et al. (2005) mention the message ordering types when multicasting messages in a distributed system, however can most of the ordering types be transferred to the system for this project except causal ordering. The orderings can be used by the clients to make it able to the server to execute the messages in the right order.

### 3.1.3 How to Adress the Difficulties

Several disadvantages and difficulties is mentioned in the above section, 3.1.2, which has to be evaluated in the upcoming system developed in this project. For a stable system
must all disadvantages be evaluated to make sure that the system acts as intended, not all disadvantages might affect the system in this project but must still be evaluated to make sure they do not.

**Security**

The system for this master’s thesis work has from the planning phase both some advantages and disadvantages regarding security. The disadvantage is the communication with the test device located at skaitevaara. All test devices communicate using plain text which can allow unauthorized users to access the device and communicate with it. The only device that has some safety is the power switch which requires a username and a password for communication, this is not perfectly safe but safer then the rest of the devices. To assure authentication a server should be put up at skaitevaara which authenticate all users connected to the test device, as the system works today the test device is connected directly to the network without any authentication control. Figure 3.4 shows how the a authentication server should be added to the device at the mountain top to assure authentication. The advantage however with the system is the science network which is a secure communication channel as it is not connected to any other network or the Ethernet. This prevents users to directly connect to the test device from the Internet, but the server added to the system is connected to the other networks at Esrange as well as the Ethernet. The server is going to have two network interfaces where one is connected to Esrange network and the other one is connected to the test device secure channel. The conclusion is that there is a possibility that a unauthorized user connects to the test device even though the chance is low, but a authentication server should be added to the system to make it even more secure.

Esrange has a number of security barriers such as firewalls to prevent users to access their internal network which gives the planned software a build in security without implementing it. There is however always a possibility for unauthorized users to get inside the internal network and access software. As the client is going to communicate with the server located at the ESS within this internal network, both authentication and confidentiality should be included in the system, this could be done by encrypting the messages sent between the client and the server as well as including an authentication such as username and password to be able to communicate with the server. The encryption assures confidentiality and the username and password assures authentication. The authentication mechanism can be built into the software to simplify the usage of the software, but the disadvantage is that if an unauthorized user gets hold of the software the authentication is not worth anything. The security compared to the usage simplicity is a choice Esrange Satellite Station must take.

The conclusions from the study of the articles about security are that both authentication and confidentiality should be implemented in the system to assure security. An authentication server at skaitevaara would be the optimal solution but it is unlikely to become reality, the security aspects between the client and the server should however be implemented.

**Scalability**

Scalability is not the biggest concern in this system as the server works almost as a communication portal between the test device and the clients. As the number of users are low and the variables stored in the server is close to constant, space, space-time and structural scalability are of no concern in the system.
Load scalability is an property that can become an issue in the system if not designed correctly. There are two things in the system for this project that needs to be designed properly to assure load scalability is maintained; (1) the server should not send too many messages within the system to remain effective during heavy loads of clients and (2) the messages within the system should be concise to reduce the time to send and receive them.

To maintain efficient during heavy loads of users in the system is not crustal at the moments, however is it good to implement such a solution if the conditions changes. Operators and technician are usually the only ones using the system, but other clients should be able to connect to the system to make changes or overview the system status. If the server sends a lot of unnecessary messages e.g. when clients connect, disconnect can the number of messages become large when the number of clients increases. An example of a bad implementation is when one client connects to the system, it sends a connect message to all other clients which then will send a alive message to all clients to make sure everyone know it is alive. Then the number of messages sent in a system when a clients connects is \((x \times (x - 1)) + x\) where \(x\) is the number of clients in the system before the new client connects, this means that for ten clients would the system send 100 messages and for 50 clients 2500 messages. This design is not load scalable for large amount of clients. In the system for this project the server is the only one who needs to know about all clients in the system, but never less the number of messages in the system e.g. status messages must be short and effective to make the system scale well.

Figure 3.4: A design where authentication would be possible.
with multiple users as stated in case two. Status messages in the system should be sent once every second to keep the clients up to date with the server status, this makes the server a bottleneck as there is a limit of how many clients one can send a message to during one second. If the number of users reaches over this limit, a second server could be added to the system to provide as a backup during heavy loads of clients. Most computers use several cores which means that several processes can be executing on the same server at the same time. This enables the server to perform multiple actions at the same time and remains more effective, the server should be implemented by using threaded processes.

The conclusion about scalability is that only one scalability, load scalability, effects the outcome of the efficiency in the system in this project and even that has little effect. However, is it a good idea to implement a system which supports a lot of users if the conditions changes.

Time

In this system physical time is crucial as macros sent by OASIS is executed at specific time, if the client time settings is not accurate with the servers the operators will not have a good overview when the macros will be executed. As the local physical time at the client is not important the best way to implement this is to have the server distribute the time in the status messages. The problem mentioned in the articles is that the clocks at the client will not be perfectly synchronized with the servers which can cause problems, however this only applies to big distributed system communicating over the internet. This project’s system will communicate within the local network at the satellite station at Esrange which will result in small network delays and enables the system to use physical time.

As several clients are allowed to be connected to the server at the same time should some sort of ”happen before” relationship be added to the system design, however is vector clocks not suitable in the system as the clients do not communicate with each other. Lamport’s logical clock, Lamport (1978), is good to implement in the system design to determine the ”happen before” ordering within a single client. Lamport logical clock involve all processes in the distributed system, but in this system it is a good idea if the client has a logical clock which the server uses to order messages. This usage will be discussed more in the message ordering section below.

Message Ordering

In a system where clients do not communicate with each other and only with the server is causal ordering hard to implement as the design assume that each message is multicasted through out the system. Total ordering can be used in this project to make sure that each message is processed by the server in the order it was created among the clients by adding a sequencer to the system, but a closer look to the design reveals that a sequencer is not necessary. Figure 3.5 shows how a sequencer could be added to the system, as the first client sends a request to the sequencer which response with a sequence number and the other client does the same after the first. The clients then send the messages to the server with the given sequence number which executes the messages in the right order. As the clients do not communicate with each other and only one single unit executes the messages is the sequencer unnecessary because if the messages are sent to the server directly the server itself becomes a sequencer as it processes the messages in the order they arrive. If a sequencer was added to the system, the order of the messages executed
is the order the request arrives to the sequencer which is the same order the messages would have been executed if it was sent directly to the server. As the communication between the clients and the server are using TCP, no messages will be lost.

As discussed in the last section about time, is it good to have a logical clock in each client to assure that each message from one single client is processed in the correct order and the system message ordering will then be FIFO. As the server uses threads in the solution messages can be executed simultaneous which can be problematic if the message is from the same client, then it is good if the server does not execute messages from the same client simultaneous and instead waits until the message before is done executing.

Conclusions about message ordering are that FIFO is the best choice in this system and should be used. As both the server and the clients are located at the same network at ESS can message ordering have a low priority as messages will not be lost and will come in the right order at almost every time. As users have to click a button or some other graphical element to send a message to the server and the messages are transferred fast on the network will the simultaneous message executions arise very rarely. One solution that makes it possible to use a non-ordered design is if only one client can be in control of changing parameters, but this give birth to several new problems such as who decide which client is in control and what can be done if another client wants that control.

### 3.2 Distributed Systems Interacting with Test Devices

There are many articles describing distributed systems which interacts with test devices of different kinds and one of them is Grimaldi and Marinov (2001) which suggest usage of an object oriented programming language such as C++ or Java to build this kinds of systems. He proposes a client-server solution where the server, which communicates with the instruments, is a web server and the user can access the service using a web browser. The interaction with the server would go through a web applet and the communication between the client and the server could be implemented using Java RMI or COBRA. They mention advantages with web services such as interaction with other distributed systems and an opportunity to use the mobile phone to interact with the server.
According to Kalaizakis et al. (2003) are most distributed measurement system based on the client-server paradigm where the server provides services to change and monitors test devices and the clients allowing some user interface to the users. They have developed a design for RES, Renewable Energy Source, system which includes several data collection servers connected to one server providing information service to clients. This system is designed as a three-tier system structure as the many parts of the system combined forms a very complex system.

Both Pianegiani et al. (2004) and Alegria and Ramos (1997) as well as Arpaia et al. (2000) has developed designs for distributed measurement systems and they all state the same that client-server is the best system structure. Some solutions are more complex than others and some are simpler, but the basic function is that there is a server which provides clients with information. How the underlying system structure between the server and the test equipment is developed is not important in this study as in this project there is only one collection of test devices located at the same physical place. Arpaia et al. (2000) for example has several servers where the underlying servers communicates with test devices located at physically different places and then sends that information to a centralized server which provides information to the clients.

Conclusions from these articles can be drawn that some kind of client-server architecture is the suitable design choice, if the server provides service through a ordinary client-server architecture or a web service is a choice which must be made together with ESS when more details about the projects has been worked out.
Chapter 4

Method

This chapter describes how the work was done and which methods that was used. This master’s thesis project stretches over 20 weeks in total and a detailed week by week plan can be found in appendix B.

4.1 Preliminaries

This master’s thesis project started with an introduction to the problem as well as the existing software and the test devices. Documentation for each test device as well as for the Skait Control was given. The code for Skait Control was given because not all commands were available through the documents. The antenna switch, which was built by Esrange employees, does not have a complete documentation and a lot of information has to be found by looking at the old code. OASIS has a well documented guide about how to communicate with it.

4.2 How the work was done

The project was splitted in three major parts; planning, implementation / testing and documentation. The first part is the planning phase which started with the introduction to the problem and to existing software, Skait Control. The test devices were brought down from the mountain to be put at the satellite station for easier testing. All of the documentation for the test devices where read and some tests where made using telnet to make sure everything worked. Using telnet gives an opportunity to interact with the test devices without having to develop a application to accomplish that.

The next part in the planning process was to figure out how to solve the problem given. The first thing that was done was to find articles about how others have solved similar problems and how they had structured their systems. With the requirements from ESS and the architectural structures proposed in the articles was the choice to make an client-server architecture. As a request from ESS, the client was developed in TCL/TK and the server in C++. The next step in the process was to, with the guidance from the articles, design the basic functions in the server and in the client. The server’s core functions were split into several different files to give the server a good structure. The result from the planning of the structure of the server ended up looking like figure 4.1.
Figure 4.1: How the structure of the server was planned to look like.

`GUILListner` and `OASISListner` is receiving and sending messages to, respectively, the client and the OASIS system. They both forward their messages to the `MessageDistributor` who take care of the messages depending on their content. The `MessageDistributor` is the core in the server and everything that happens goes through there, it is in control of macro lists from OASIS, disable and enable the matrix, sends commands to the test devices, sends status messages and a lot more, see section ”The developed system”. `CTTMsgConverter`, `SGMsgConverter` and `SwitchMsgConverter` receives messages from `TestDeviceSender` and `MessageDistributor`, converts them and forward them. `TestDeviceSender` receives and sends messages to the different test devices.

The next part in the process was the implementation part where the actual software
was implemented. This phase started with the creation of the server where all necessary
files, designed in the planning phase, was created. The communication between the
server and the client started when the client had a basic graphical structure. The
development process followed the basic schema below when new features was added to
the software.

1. Develop the feature in the server, adding all the new methods needed and variables
to save information.

2. Develop a graphical interface in the client, with buttons, spinboxes and other
graphical elements to make changes and labels to display variables.

3. Add message functions into the system at both the client and the server to make
them able to communicate. The server must understand messages from the client
about the new features and the client must understand the new status messages
from the server.

4. Test the new function and fix all errors found within the system.

When all the client-server-test device interaction were working the development be-
gun with the communication between the server and OASIS. OASIS has document
specifications regarding the communication which makes it easy to communicate but
hard to make individual changes within. The reason OASIS has such strict communi-
cation specifications is because it communicates with a lot of servers on the same UDP
socket which makes changes from OASIS side almost impossible.

The project followed the week by week plan in appendix B most of the time.
Chapter 5

Results

The result of this master’s thesis project is a developed system based on the requirement specification, appendix A, and the information gathered in chapter three. This chapter will describe the result of the project together with an discussion based on the articles in chapter three. First is a description of the test devices and how it changed during this project.

5.1 Test Device

Since the start of this project some changes have been made among the devices at skaitevaara. The managed power outlet was added to the devices to get a control over the power supply to the rest of the devices and the anritsu devices was changed to another signal generator. The anritsu device broke during the project and a new one was bought to replace it.

Figure 5.1 shows an overview of the test devices located at the Skaitevaara and it is all a part of the device for testing the antennas located within the Esrange area. The solid lines between the equipment are communication cables and the dotted lines are power supply cables. The Network switch in figure 5.1 is an ordinary network switch which receives messages from the science network and distributes it to the correct device and the other way around. The Science network connected to the switch is the network from the main building used by the client-server system.

There are two Ethernet/serial converters in the system. Both the CTT and the Antenna switch has serial ports for communication and the messages sent to them uses the TCP/IP protocol through the network. This means that a converter has to convert from TCP/IP to serial and vice versa.

To be able to reboot devices remotely a Managed power outlet has been added to the equipment on Skaitevaara. A feature to remotely reboot a device is desirable from the ESS point of view because the devices are located so far from the actual station, to reach the mountain top one must either use helicopter or a snow mobile. The power device has eight power outlets in that every device is plugged into, a user can turn a single outlet on and off using the client software. The managed power outlet receives messages via TCP/IP and sends power status messages back.

The CTT, Collimation transmitter, is a signal transmitter that generates a X-band signal with a frequency of 8153 MHz which is distributed through the antenna switch to one of the transmitting antennas. The output level of the CTT is adjustable by step of
Figure 5.1: An overview of the test devices.

1 dB in a range from -71 to +10 dBm. The CTT has a doppler which can be turned on or off, the doppler shifts the frequency with a range from -300 kHz to +300 kHz at the rate of 1.7 kHz/s. The doppler can be used to simulate a signal from a satellite where the frequency changes because of the movement, called the doppler effect.

The Signal Generator together with the CTT are the two units in the test device system that generates signals that is distributed to the antennas. The Signal Generator, unlike the CTT, has many features and variables. It can generate every frequency in the S-band and X-band span and it can have the output level from -135 dBm to +14 dBm. The unit has a graphical web interface, see figure 5.2, which looks exactly like the front of the unit, this can and will be used by the operators to change parameters not commonly used. The software developed during this project has only features to change the frequency and the output level as well as turning the transmitter on and off.

The last unit among the test devices is the Antenna switch made by employees at Esrange Satellite Station. It receives transmitting signals from either the CTT or
Signal Generator and forwards it to a small antenna located at Skaitevaar. There are several small antennas located at the mountain which points at different big antennas. The big antennas have two small antennas pointed at them, one for S-band and one for X-band. The Antenna switch is built to handle up to eight small antennas, four S-band and four X-band, but only five are used at this moment. The switch receives commands from a serial port and the operators can change which antenna to transmit to by sending commands via the developed software. To make the switch commands serial a Ethernet/serial converter is used.

5.2 The Developed System

The system developed for this master’s thesis project was designed based on two things; the requirement specification and the design principles found in articles summarized in chapter three. The fundamental system structure choice is a client-server structure, see figure 5.3, because this kind of structure is well known at ESS, tested over many years and fulfills all the requirements in the requirement specification. The benefit of using a client-server structure is if ESS later on want to develop a new graphical interface for the system, they do not have to change anything in the underlying structure of the server or vice versa. This is an advantage because less code has to be re-developed which is time saving.

![Figure 5.3: Basic structure of one server connected to several clients.](image)

The rest of this chapter is splitted into two main sections, the server and the client. Both parts of the system are described, how certain problems were solved as well as discussion based on the design and the articles.

5.2.1 Server

The server can be divided in to three parts, the one communicating with the clients, the one communicating with the test devices and the one communicating with OASIS. In the middle of these three parts is the Message distributor which coordinates the messages in the server and keep tracks of the system variables. Figure 5.4 shows how the different parts of the server is connected.
Test Device Communicators

The test device communication parts consists of CTTCommunicator, PowerSupplyCommunicator, SignalGeneratorCommunicator and SwichCommunicator. As the name indicate does CTTCommunicator communicates with the CTT device, the PowerSupplyCommunicator with the managed power outlet, the SignalGeneratorCommunicator with the signal generator and the SwichCommunicator with the antenna switch. All of these have much in common, all of them establish a TCP socket connection between themselves and respective device, they receives messages from the device and converts it to a specific format which is sent to the MessageDistributer. All test device communicator classes receives messages from the MessageDistributer which they forward through the TCP socket to the test device, the most common message is a status request message which is sent with a frequency of one Hz (one message per second). The status request messages looks different for every device, and the devices returns different messages. Each class translates the messages and sends them to the MessageDistributer.

Below is an example of how the status message call and return looks like, to understand both the CTT and the antenna switch returns message, which is in ASCII, must they be translated into binary. The signal generator does not have a general status call because it has a lot of variables, instead there is a status call for every variable, the FREQ? returns the current frequency in Hz, the POW? returns the current output level in
5.2. The Developed System

dBm and the OUTP? returns 1 if the signal generator is transmitting signals and 0 if it does not.

CTT status message call:
TS?
CTT status return message:
CT8;<0

Switch status message call:
$016
Switch status return message:
!2D7400

Signal generator message call:
FREQ?
POW?
OUTP?
Signal generator return message:
+3.0840000000000E+09
-4.00000000E+001
0

The CTT status return messages for example have ASCII to give true or false for variables using their binary number, except for the first two, 'CT', which are indicators that the message is a status message. Only the last four binary numbers is used and if the third number is '8', which is 0011 1000 in binary, then the '1' indicates if the modulation is on or off, the three '0' is used to display if there is any of three alarm. If the number is '0', like in this example, then there is no alarm. Translating all of the ASCII characters will give the current status of all settings in the CTT device.

MessageDistributor, Time and Main

The Main component contains just one function which starts the server by calling the MessageDistributor. The Time component is used by the MessageDistributor to give the time and date at a specific moment in a specific format, it takes care of most of the time related functionality in the system like determine if a macro should be executed, put on queue or if it is too old to be executed and therefore thrown away.

The largest and most important part in the server is the MessageDistributor. All the core functionality in the MessageDistributor is listed below.

- Starts all the other components in the system in a new thread.
- Controls and stores the client local or remote status.
- Controls and stores the macro list enable or disable status.
- Stores all test devices statuses in different classes.
- Creates client and OASIS status messages and distributes them at a rate of one message per second.
- Creates messages to client containing OASIS macro list or antenna names. This is sent during the start or if the macro list changes.
Chapter 5. Results

- Stores OASIS macro list.
- Keeps track of when to execute the next OASIS macro in queue.
- Executes OASIS macro.
- Change the parameters in a macro command.
- Collects information from several .txt files.

The MessageDistributor is the first, and only, part in the server started by the Main. The first thing the MessageDistributor does is starting all the other components in the system in new threads, the threads makes it possible to listen to multiple sockets at the same time as well as performing different actions at the same time in different parts of the server. The second thing it does is reading client macro names together with corresponding macro variables, OASIS macro names together with corresponding variables as well as antenna names. This information is stored into text-files to make it easy for an operator or system administrator to change if necessary. The client macros are used to set up and start the signal generator using the client and they looks like the following:

```
Spot4_X-band
8153
-2.5
Spot5_S-band
2218
-11
```

All macros are stored in the same file and each macro has three parameters stored in three lines. The first line is the macro name, the second is the frequency in MHz and the last one is the output level. Macros are stated after each other without any spaces between them. There are two ways of changing the frequency or the output level in the macros, the first is to do it using the GUI which has such a feature and the second is to change the file manually using a editor. If the server is running and the macro file is changed manually, a user can update the list using a feature in the client.

The status message sent to the clients has the following appearance:

```
time!2010-02-09 10:15:48
status!111110001014011111103
SG!+8.0500000000000E+09!-2.00000000E+01!0!
```

The first part of the status message is the server’s time, recognized by the prefix 'time!' in the beginning, it tells the client what time to display in the client. The local physical time in the client is not important and are provided by the server because the important thing is to know when the server is about to execute a macro. The time offset between server and client created by the time it takes to create, send and receive the status message, discussed in the last chapter, is not important in this project. Both the server and the clients are assumed to be on the same local area network, LAN, at the ESS which will make the time offset insignificant.

The second part of the message is the actual server status, recognized by the prefix 'status!' in the beginning. Most part of the system states are given as '0' or '1', where zero is off and one is on. There are only two statuses which are different, the CTT output
level which can be any number from 0 to 71 and which antenna the antenna switch is connected to, this value can be any number between 0 and 8. The last number, which is antenna connection, corresponds to the file read earlier and if the number is one then it is connected to the antenna at position one in the file, zero means that the switch is not connected to any antenna. The last part of the status message is the signal generator status, it is recognized by the prefix 'SG!', and contains three values separated with an '!'. The first value is the frequency, the second the output level and the third is '1' if the signal generator is transmitting a signal to the antenna switch and '0' if it does not.

The MessageDistributor stores the OASIS macro list in a queue ordered by time for execution. The first macro in the queue is controlled each second to check if the macro should be executed, the time is taken out and sent to a function in Time which checks if the macro is too old to be executed. If a macro is older then ten seconds compared to the server’s physical time will be removed from the queue and not executed, this happens if the macro list has been disabled or if the macro list was just read and there were old macros in it. If the macro is not too old to be executed a new function in Time will be invoked which returns true or false if the macro should be executed at this time. If it should, the MessageDistributor finds the corresponding macro name in the OASIS macro file and execute all commands in it. An example of the macro list looks like the following:

```
2010-02-09 11:52:30 PrepareMission CTT_example ESX
2010-02-09 11:53:30 Start_XB
2010-02-09 12:15:00 Stop_XB
2010-02-09 12:23:00 PrepareMission SG_example ESX
```

And a macro itself looks like:

```
Mission: CTT_example
Band: X
CTT_modulation
CTT_transmitter
CTT_doppler
CTT_output_level 5
EndMission: CTT_example
```

The OASIS macro list contains the date and time for the execution, what type of macro it is, the name of the macro and a antenna name. There are a few different types of macros, the most common is 'PrepareMission' which states that a mission should be prepared for but not executed. The last parameter which is the antenna name is optional, the antenna switch switches to the given antenna if specified. All commands in the macro is set up in the test devices to be prepared for the next set of macro types which is start missions. They commands the prepared test device to start transmitting
signals until a stop mission is executed. The last type of macro types is 'ENABLE' and 'DISABLE' which enables and disables the macro list.

In conclusion, the MessageDistributer is the "spider in the web" which supervise all other parts in the system.

Client and OASIS Communicators

The GUIListener and the OASISListener is the last parts in the server and they have much in common with the test device communicators. The GUIListener creates a TCP socket for each client connected and saves them in a list, the limit of clients is set to ten. If a client disconnects or closes the socket the corresponding position in the list are removed and the server does not try to reconnect to a lost client. Messages to the client is sent using the GUIListener which iterates through the list of clients and distributes the message, the messages from the client is received and forwarded to the MessageDistributer and no distinction is made on which client it came from.

The OASISListener communicates with OASIS using UDP and predefined messages, this is a requirement from the OASIS specification to eliminate unnecessary work for the OASIS developer. OASIS and OASISListener communicates with three different UDP sockets, one to receive commands from the OASIS, one for server to OASIS messages and the last for server to OASIS status messages. When OASIS sends a 'Get_matrix_file' message together with a file name parameter, the server must retrieve a new macro list file from OASIS using FTP.

There are several ways to download a file using FTP and the choice was made to use a tcl/tk package called Expect. The tcl/tk script was saved as a separate file and invoked using a system call as follow

```c
string siteAndPlace = FTP_IP;
siteAndPlace += ";";
siteAndPlace += filename;

string call = "./ftp.tcl ";
call += siteAndPlace;
const char * realCall = call.c_str();
system( realCall );
```

The script is invoked together with the FTP server’s ip-adress and the filename. It is simple to use Expect and it takes very little effort to make it work. The core of the script looks like the following.

```c
class Export {
    virtual void run() {
        string site = "FTP_SERVER_IP";
        string command = "./ftp.tcl ";
        command += site;
        system(command.c_str());
    }
};
```

The script is one of the ways to download a file using FTP. There are several ways to download a file using FTP and the choice was made to use a tcl/tk package called Expect. The tcl/tk script was saved as a separate file and invoked using a system call as follow

```c
string siteAndPlace = FTP_IP;
siteAndPlace += ";";
siteAndPlace += filename;

string call = "./ftp.tcl ";
call += siteAndPlace;
const char * realCall = call.c_str();
system( realCall );
```

The script is invoked together with the FTP server’s ip-adress and the filename. It is simple to use Expect and it takes very little effort to make it work. The core of the script looks like the following.

```c
spun ftp $site
expect "Name*:" { exp_send "example_user_name\r"
} timeout {
    puts "Timed out, try later!"
    exit
}
expect "Password*:"
exp_send "$password\r"
expect "failed" {
    puts "Bad luck, failed!"
    exit
```
5.2. The Developed System

```tcl
} "ok" {
    puts "Logged in!"
}
expect "ftp>"
exp_send "get $file $localfile\r"
expect "ftp>"
exp_send "exit\r"
```

The `Expect` package has three commands used in the script above and that is 'spawn', 'expect' and 'exp_send'. The spawn command connects to the FTP server given as parameter to the script. The expect command recieves messages and checks if the message is the same as expected, if it is it continue with the script or else it will generate an error and exit. The expect command can have several option like when the password is sent, it can either return "failed" which means that the password is wrong or "ok" which means that the password is correct and the script is logged in. The exp_send message sends a message to the FTP server.

Many parts of the server has similar designs and works very similar. The design and implementation choices are discussed in the next chapter.

### 5.2.2 Client

The client is developed in the program language tcl/tk, unlike the server which is developed in C++. The client graphical interface is designed with four major windows which all has its own area of responsibility, three of the windows are designed as tabs and are not always visible, the last one is always displayed. Figure 5.5 shows the first tab of the client which is displayed when the user starts the client application.

![Figure 5.5: The first tab in the client GUI.](image)

The window that is always displayed is the information window to the right, the top
has information about the important parts of the system and the bottom displays the connection between the antenna switch and the antennas. In this picture is the CTT connected to ETX X and the signal generator to ETX S. The first tab, to the left in the GUI, displays if the server is in local or remote mode, if the macro list is enabled or disabled as well as the complete macro list.

Figure 5.6 shows how the second tab is designed and it is used to display the options and errors in the CTT. The top left part of the window is used to display to the user which features in the CTT that are active and also allow the user to turn the feature on and off. The bottom left part is used to display and alarm the user if there are any errors within the device. The right square makes it possible for the user to change the output level.

The last tab, see figure 5.7, is the signal generator tab and it is used to change the settings in the signal generator. There are several buttons to the left of the window which will open different windows in the right square allowing the user to execute macros, create new macros and update an already existing macro. The current settings of the signal generator is displayed in the bottom and there is also a possibility to turn off the transmitter. The first button starts a web browser which will open the graphical web interface for the signal generator as in figure 5.2.

The client does not use threads like the server to support several tasks to be performed simultaneously, event listeners are used instead much like the ones used in Java. Both graphical elements such as buttons and non graphical like sockets uses this kind of listeners which invokes functions upon changes. Below is an example of how such a listener is added when a button is created.

```
button .c.exampleButton -background gray -text "Example" \
    -command { Example_Button_Clicked }
place .c.exampleButton -x 0 -y 0 -w 60 -h 25
```
5.2. The Developed System

5.2.3 Discussion

The conclusions drawn from the articles in chapter three was evaluated and prioritized when the system design process begun to get an idea of which properties are important and which are not. Some of the design proposals in chapter three was not implemented as it was prioritized low and was put into future work as the project time was not enough to implement it. As the security is high here at Esrange there is much troubles for an unauthorized user to enter the network was the prioritization for a encrypted communication between the client and the server low. There are several more desirable softwares at Esrange to hack into if you are a unauthorized user which is why the chance that someone hacks into this project’s system is low. The encryption was never implemented in the system as the time was not enough and is put as future work.

As both client and server are located at the same local network is the delays in the system infinite small which made message ordering low prioritized as well. It was not implemented but the implementation of FIFO, if needed, in the system does not require much work. It is however now put as future work until the system has been tested live and an evaluation can conclude if it is needed.

The server is implemented to scale with several users, however is there a limit set to ten users at the moment. The only ones supposed to use the system is the operators and technician which will use one client, the limit of ten clients at the same time will probably never be reached when the system is live. The system should scale well with several users as there is no complex solution involving the number of users and the limited number of users in the system is the number of status messages possible to send during one second in the network. The status messages consists mostly of zeroes and ones which displays true or false to minimize the size of the messages. This to allow better scaling in the system.

One design issue with today’s solution is that the system does not keep the clients...
apart which means that the server does not know which client it got a message from. This can become problematic if several clients are connected to the server and changing parameters. This can be solved if just one client can set the server in local mode, but the problem is if other clients want to enter the local mode while the first is still in there. This can result in a lock in the system and has to be worked around in some way to be solved.

The original server design described in the method chapter, chapter four, figure 4.1 and the outcome of the server in the end, figure 5.4, does look alike but has some differences. The time class was not in the designing phase because it was intended to be included in the MessageDistributor class, however did the class became too complex which resulted in a breakdown of the class into two classes. The original design had a class called TestDeviceSender which purpose was to communicate with all test devices, but after further investigation was it concluded that the communication with the test devices was done in such a different way that the TestDeviceSender class would have been hard to implement. Instead each test device class took over the communication which has both advantages and disadvantages. The advantages is that each class can specify its communication based on the specific test device and the disadvantage is that much code is similar in each class which is unnecessary to have. If the project would have been done with less limited time the design choice would be to implement the communication as one class, but with the time given was the choice to make each class communicate with its own device to minimize the chance of complexity problems.

To collect files from OASIS using FTP the server uses a TCL/Tk script instead of an package in c++ and the reason for this is that other applications at ESS uses TCL/Tk scripts for FTP which will allow employees to understand what is done in the code after the software has been handed over to them. The other reason is that TCL/Tk is installed in the system which is not the case with the FTP C++ package, if the C++ package had been used it had to be reinstalled if the server changes hardware which can cause problems for the system administrators.

As the anritsu device broke and a new signal generator was bought was some time lost in the implementation phase which resulted in that not all signal generator variables was added to the system as was planned in the beginning. The variables can however be changed using the graphical web interface which can be opened from the client. In future work can more variables be added to make more use of the client and less of the web interface.
Chapter 6

Conclusions

In this paper, a client-server architecture has been described that controls antenna test device instrumentation within the Esrange Satellite Station own network. The description includes the result of the project as well as background information and a in-depth study about similar projects. The resulted client-server system includes one server which communicate with the test devices and then allows one or several clients to monitor and control the test device through the server. OASIS is an automated system included in the system to automate tests to be executed without any supervising from the satellite operators which will be the main users of this system.

The achievement for this project was to create a fully working system to control the test device which could go live after the ending of this project. This main goal has been achieved as well as most of the requirements in the system requirement specification in appendix A. The only goal that was not fulfilled was the time goal because the project did take more time than planned and the biggest reason for this is that the anritsu device broke. When the device was replaced was three to four weeks of code lost which had to be thrown away and re-coded. The loss of the device had an impact on the system complexity too as many variables and functions in both the client and the server was named after the anritsu. Some of those could still be used but had to be renamed to something, e.g. signalGenerator or SG, to be understandable for members not part of the project. Except for the broken anritsu has the project suffer no drawbacks.

6.1 Limitations

There are not any major limitations in the system today because the system used before this project, Skaute Control, was evaluated and the limitations from that software was solved and added to this project’s system. The system for this project did take all benefits from the old system and redesigned the disadvantages to get rid of the limitations. However, as the project would have needed some more time to be complete, some limitations still exists in the system.

As the system is implemented today, the client does not know if a test device has disconnected. This can result in confusions for the users as they try to communicate with the test device without knowing that the server cannot connect to it. This can be solved changing the colour of the specific test device when it is disconnected to alert the operators that the device is unconnectable.

The server does not keep track of which client is sending a message to it and several
clients can change system status at the same time in the system if the server is set to local. This is not good if several independent users try to use the system at the same time and changes settings on the same device. A solution to this is to only allow one client to be in local mode, the problem is to solve how to decide who can be in local and how to change local mode among clients.

The server cannot change all variables in the signal generator as planned from the beginning which is a limitation. The system lack the opportunity to turn the transmission of signals on in the signal generator, it can only be turned off. The client can not set any modulation settings in the signal generator which should be needed as some of the antenna tests uses is.

Many of these limitations are future works which should be implemented later into the system.

6.2 Future Work

As the main goals of this project is achieved is there still some future work that can be done to get rid of the limitations in the system. The first thing that should be done is to make it visible in the client if any test device has disconnected. The second thing that has to be done is to make the client able to start the signal generators transmitter and change the modulation settings.

Only one client should be able to be in local mode in the future to prevent several users from operating in the system simultaneous. The system should also implement the suggestions made in chapter three concerning encryption and authentification.

All these future work should be done in the future but is not necessary for the system to go live.
Chapter 7

Acknowledgements

First of all I would like thank to my supervisor, Christer Jonsson, at Esrange for his advice and support through the whole process. I would also like to thank Esrange for allowing me to do my Master’s Thesis work there and all the co-workers who have helped me when I had problems. I would like to thank my supervisor at the Department of Computing Science at Umeå University, Jonny Pettersson, who, with his great experience, has guided me through the writing process of this project.

A special thanks to my family without whose support and encouragement this master’s thesis work would have been impossible.

Last I want to dedicate this work to my children, Saga and Älwa Stålnacke.
References


Appendix A

System Requirements

1. The system must have a client application.
   (a) The client must be developed in TCL/TK.
   (b) The client must be able to send CTT commands to the server.
   (c) The client must be able to send signal generator commands to the server.
   (d) The client must be able to send power switch commands to the server.
   (e) The client must be able to send antenna switch commands to the server.
   (f) The client must be able to receive status messages from the server.
   (g) The client must be able to display system status.
   (h) The client must be able to set the application in remote or local mode.
   (i) The client must be able to disable and enable the OASIS macro list.
   (j) The client must display the time that the server sends to it.
   (k) The client should not display something as done until it has received a status message about it, for example disable the matrix.
   (l) The client should have options for all the signal generators variables.

2. The system must have a server application.
   (a) The server must be developed in C++.
   (b) The server must use TCP/IP connection to the test equipment.
   (c) The server should be able to handle multiple clients.
   (d) The server must be able to correctly handle OASIS macros.
   (e) The server must not execute macro commands that are older than 10 seconds.
   (f) If a macro time occurs and one macro is already executing, the server must save the macro and execute it when the first is done.
   (g) The server must be able to reconnect to a test device if the connection is lost.
   (h) The server must be able to get files through FTP from OASIS.
   (i) The server must be able to send and receive messages.
       i. The server must be able to send commands to the CTT device.
ii. The server must be able to receive messages from the CTT device.
iii. The server must be able to send commands to the signal generator.
iv. The server must be able to receive messages from the signal generator.
v. The server must be able to send messages to the antenna switch.
vi. The server must be able to receive messages from the antenna switch.
vii. The server must be able to send commands to the power switch.
viii. The server must be able to receive messages from the power switch.
ix. The server must send status messages to the client and OASIS.
x. The server must be able to receive messages from the client.
xi. The server must be able to receive messages from OASIS.

3. The system must be able to handle OASIS macrolist correctly.
Appendix B

Planning

1. Week 1
   (a) Start up the project.
   (b) Go to meetings to get information about what to do.
   (c) Start searching for articles about similar projects.
   (d) Start sketching a rough server application structure.

2. Week 2
   (a) Read the articles about similar projects.
   (b) Write down those things that can help in this project.
   (c) Start up with the final report by writing the goal part.

3. Week 3
   (a) Start implementing the server by making classes and write the name of the functions in the classes from the sketching done earlier.
   (b) Start learning about TCL/TK which is a new program language for the master’s thesis student.
   (c) Write more in the final report, writing the goal part finish.

4. Week 4
   (a) Make the information tab, which displays all status information, in the client.
   (b) Fill in the code in the already existing methods in the server.

5. Week 5
   (a) Make it possible for the client and the server to communicate.
   (b) Get the server to start send time messages to the client.
   (c) Keep on building the client information tab.

6. Week 6
   (a) Make it possible for the client to graphically disable and enable the matrix.
(b) Make it possible for the client to graphically set local or remote.
(c) Implement so the server can change from local to remote.
(d) Implement so the server can disable or enable the matrix.
(e) Make the server start sending status messages.

7. Week 7
   (a) Read about the CTT and how to communicate with it.
   (b) Implement the communication between the CTT and the server.
   (c) Create the CTT tab in the client application.
   (d) Implement so the client can send CTT commands to the server.

8. Week 8
   (a) Implement so the client can send CTT commands to the server.
   (b) Add the CTT status to the status messages the server is sending.
   (c) Make the client understand the new status messages and set the status according to it.

9. Week 9
   (a) Implement the communication between the power supply and the server.
   (b) Implement the graphical part in the client for the power supply.
   (c) Implement so the client has a confirmation frame if anyone wants to turn off the power to an equipment.
   (d) Add power supply status to status message from server.
   (e) Make the client understand the new status messages and handle it.

10. Week 10
   (a) Read the programming interface specification for the switch.
   (b) Implement the switch communication.
   (c) Implement the graphical part for the switch in the client.

11. Week 11
   (a) Add switch status to the status messages from the server.
   (b) Make the client understand the new status messages and handle it.
   (c) Read about the signal generator device.
   (d) Start with the signal generator tab in the client.

12. Week 12
   (a) Finish with the signal generator tab in the client.
   (b) Implement the communication between the server and the signal generator.
   (c) Add signal generator status to the status message from the server.
   (d) Make the client understand the new status messages and handle it.
13. Week 13
   (a) Together with the OASIS staff, make the macros necessary for the server and OASIS to communicate.
   (b) Implement macros in the server, so it can handle them correctly.

14. Week 14
   (a) Find more articles for the deep study.
   (b) Read the articles and order them to be able to make a good structured deep study.

15. Week 15
   (a) Writing the final report section problem description and the deep study part.

16. Week 16
   (a) Finish the deep study and start with the accomplishment.

17. Week 17
   (a) Write the result and conclusions.
   (b) Go through the whole report and fix all the errors and misspellings.
   (c) Send in report to internal supervisor.

18. Week 18
   (a) Fix the final errors.

19. Week 19
   (a) Make the presentation for Esrange and for the university.
   (b) Give a presentation to Esrange.

20. Week 20
   (a) Give a presentation to the university.