Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Analysis and implementation of remote support for ESAB's welding systems

- using WeldPoint and web services

by

Rickard Öh

LIU-IDA/LITH-EX-G--09/006--SE

2009-06-10

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Final Thesis

Analysis and implementation of remote support for ESAB's welding systems

- using WeldPoint and web services

by

Rickard Öh

LIU-IDA/LITH-EX-G--09/006--SE

2009-06-29

Supervisor: Johan Strand
Research and Development Department, ESAB

Examiner: Juha Takkinen
IDA, Linköping University

Abstract

This thesis was written on behalf of ESAB Research and Development department, in Laxå Sweden. One of ESAB's product areas is developing various welding systems.

Today if ESAB's customers experience a problem with one of their welding systems they call ESAB's service center. If the problem seems to have been caused by software, or if it requires log files to be analyzed, ESAB needs a way to get this system information from the customer's welding system to ESAB's employees.

One of the goals with this project thesis was to perform an analysis answering how the system information should be sent, stored and what unit in the customer's welding system that should send it. Another goal was to implement the solution that the analysis presented.

The analysis shows that WeldPoint in combination with a web service is the best way to send the system information from the customer's welding system. WeldPoint is a PC control and log software connected to the customer's welding system. A web service provides a service interface enabling clients to interact with a web server. Clients communicate with the web service using HTTP, this means that clients can easily communicate across firewalls and other network obstacles.

The thesis work resulted in three different applications written in C#.NET. The first application is a simple form called WeldPoint Remote Support (WRS). This form extracts customer information, welding system information and log files from the customer and the customer's welding system. All this information is called a case. The case is received by ESAB using the second application, WeldPoint Web service (WWS). WWS stores the received case in a database. The third application is called WeldPoint Remote Support Center (WRSC). This application is used by the ESAB employee's to view the case sent from the customer's welding system.

The above implementation has been tested and supports a robust and secure way to send and view the system information from the customer's welding system. The conclusions showed that all goals and requirements set by ESAB were met.

Abbreviations

The following abbreviations will be used in this thesis.

API	Application Programming Interface
BCNF	Boyce-Codd normal form
CAN	Controller Area Network
CPU	Central Processing Unit
FTP	File Transfer Protocol
GUI	Graphical user interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IIS	Internet Information Services
INT	Integer
MB	Megabyte
MIME	Multipurpose Internet Mail Extensions
MSSMSE	Microsoft SQL Server Management Studio Express
MTOM	Message Transmission Optimization Mechanism
PC	Personal computer
RAM	Random-access memory
RPC	Remote Procedure Call
SDRAM	Synchronous Dynamic Random Access Memory
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRAM	Static Random-access memory
SSL/TSL	Secure Sockets Layer / Transport Layer Security
TCP/IP	Transmission Control Protocol / Internet Protocol
UDDI	Universal Description, Discovery and Integration
USB	Universal Serial Bus
W3C	World Wide Web Consortium
WRS	WeldPoint Remote Support
WRSC	WeldPoint Remote Support Center
WS	Web Services
WSDL	Web Services Description Language
WSE	Web Services Enhancements
WWS	WeldPoint Web Service
XML	eXtensible Markup Language

Contents

1 INTRODUCTION 1

1.1 BACKGROUND 1
1.1.1 ESAB 1
1.2 PROBLEM 1
1.3 PURPOSE 1
1.4 METHODS 2
1.4.1 Literature study 2
1.4.2 Analysis and design 2
1.4.3 Implementation 2
1.4.4 Evaluation 3

2 CUSTOMER'S SYSTEM OVERVIEW 4

2.1 THE CUSTOMER'S WELDING SYSTEM 4
2.2 THE CUSTOMER'S INTERNAL NETWORK 6
2.3 REQUIREMENT SPECIFICATION 6
2.3.1 Goals 6
2.3.2 Requirements 7

3 ANALYSIS 8

3.1 WHAT UNIT SHOULD SEND THE INFORMATION? 8
3.1.1 $W8_2$ 8
3.1.2 WeldPoint 8
3.1.3 Standalone application 9
3.1.4 Discussion 9
3.2 HOW TO SEND THE SYSTEM INFORMATION 10
3.2.1 Email 10
3.2.2 FTP 10
3.2.3 Database via a web service 10
3.2.4 Discussion 11
3.3 HOW TO STORE THE INFORMATION 12

4 WEB SERVICES 13

4.1 HISTORY 13
4.2 THE WEB SERVICE PROTOCOL STACK 13
4.2.1 Transport and encoding 13
4.2.2 Quality of service 15
4.2.3 Description 15

5 DESIGN AND IMPLEMENTATION 17

5.1 TOOLS 17
5.1.1 C# programming language 18
5.1.2 Microsoft .NET Framework 18
5.1.3 Microsoft Visual Studio 2005 18
5.1.4 Microsoft SQL Server 2005 Express Edition 18
5.1.5 Microsoft Internet Information Services (IIS) 18
5.1.6 Web Services Enhancements (WSE) 3.0 19
5.1.7 Virtual Box 19
5.1.8 Wireshark 19
5.1.9 ZoneAlarm 19
5.2 WELDPOINT REMOTE SUPPORT (WRS) 19
5.2.1 What information should be sent? 19
5.2.2 Design 20
5.2.3 Implementation 20
5.3 DATABASE 25
5.3.1 Tables 25
5.3.2 Stored procedures 25

5.3.3 Storing the case: WeldPoint Web service (WWS) and database communication 26
5.3.4 Security 28
5.4 WELDPOINT REMOTE SUPPORT CENTER (WRSC) 28
5.4.1 Design 28
5.4.2 Implementation 29
6 TESTING 34
6.1 TESTING WELDPOINT REMOTE SUPPORT (WRS) AND WELDPOINT WEB SERVICE (WWS) 34
6.1.1 WRS form tests 34
6.1.2 WRS and WWS communication tests 35
6.2 TESTING OF THE DATABASE AND WELDPOINT WEB SERVICE (WWS) 36
6.2.1 Test: WWS sending larger data than allowed. 36
6.3 TESTING WELDPOINT REMOTE SUPPORT CENTER (WRSC) AND THE DATABASE 36
6.3.1 Test: Incoming case, one WRSC application running 36
6.3.2 Test: Incoming case, two WRSC applications running 36
6.3.3 Test: Deleting a case, two WRSC applications running 37
6.3.4 Test: Changing a case's status, two WRSC applications running 37
7 DISCUSSION AND CONCLUSIONS 38
7.1 CONCLUSIONS 38
7.2 FUTURE WORK 39
7.3 EXPERIENCES 40
BIBLIOGRAPHY 41

List of figures

Figure 1. System overview of the customer's welding system for manual welding. 4
Figure 2. System overview of the customer's welding system for mechanized welding. 5
Figure 3. A typical internal network of one of ESAB's customers. 6
Figure 4. WeldPoint Main tab. 9
Figure 5. A web service example. 11
Figure 6. The Web service protocol stack 13
Figure 7. WRS to WRSC communication example. 17
Figure 8. The WRS form. 21
Figure 9. Pop-up window for case-system information. 22
Figure 10. Messages being sent when sending a case. 23
Figure 11. Communication between WWS and the database 27
Figure 12. WSRC log- in. 29
Figure 13. The WRSC application. 30
Figure 14. The View Case file(s) menu. 30
Figure 15. The Set Case(s) as: menu 30
Figure 16. User management window of WRSC. 31
Figure 17. Simplified query notification example. 32

List of code boxes

Code box 1. Method to be called. 14

Code box 2. SOAP request message. 14
Code box 3. SOAP response message..... 15
Code box 4. WSDL 1.1 XML syntactic structure [13]. 16
Code box 5. Example of <portType> and <operation>. 16
Code box 6. Example of a simple stored procedure..... 26

Appendices

Appendix A, WeldPoint Web Service WSDL file…….……..43
Appendix B, ER diagram..…….……...……...47

1 Introduction

This chapter describes among other things the background and purpose of this Bachelor's thesis.

1.1 Background

ESAB is a world leader in production of welding consumables and equipment. One of ESAB's product areas is various welding systems. These systems can contain a welding robot, a wire feeder and power sources. These are connected to each other via a CAN bus and are controlled by different control units. CAN is a serial network standard that is mostly used in embedded systems [26].

Today if one of ESAB's customers experiences a problem with one of their welding systems they call ESAB's service center. If the problem seems to be caused by software or if it requires logs to be analyzed the problem is forwarded to the development department. A usual problem in this situation is that the service center or the developers need system information about the welding system such as: the type of equipment being used, software versions, conditions that the problem occurred in, and CAN logs.

1.1.1 ESAB

The company was founded by Oscar Kjellberg, who pioneered the development of manual metal arc welding electrodes, in Gothenburg in 1904. Since 1994 ESAB is owned by Charter International plc. Today, with operations in a large number of countries, ESAB is a world leader in production of welding consumables and equipment. ESAB operates in the following key areas: Manual welding and cutting equipment, Welding consumables and Welding automation [12].

ESAB's customers are found in the following industries: Automotive General fabrication and civil construction Pipelines, Pipe mills, Power generation, Process industry, Repair and maintenance, Shipbuilding and offshore, and Transport and mobile machinery [12].

This Bachelor's thesis has been done at ESAB's main Research and Development Center in Laxå, Sweden.

1.2 Problem

Today ESAB does not supply a simple way for its customers to send the system information (see section *1.1 Background*) from the customer's welding system to the developers at ESAB.

1.3 Purpose

The purpose of this Bachelor's thesis is to develop a solution to send the system information described in section *1.1 Background* from a customer's welding system to ESAB's employees. The system information should be sent over the internet. This solution will help the employees at ESAB's service center and development department to better analyze and understand the problem that the customer has with its system.

The main objectives with the thesis project are:

- Analyze different ways to send and receive the system information over the internet.
- Implement a solution to send and receive the system information.

- Implement two graphical user interfaces, one interface for the customer who is sending the information and one interface for the ESAB employee viewing the information.

1.4 Methods

This section describes how the work will be structured in order to satisfy the requirements and goals set in sections *2.3.1 Goals* and *2.3.2 Requirements*.

1.4.1 Literature study

First off there will be a literature study where information about the topic will be gathered. Depending on the analysis phase the literature needed will differ. For example, the implementation programming language might be different depending on the solution. In order to understand the welding system, ESAB's internal documentation will be studied. I will also ask questions and discuss different problems with my supervisor and other ESAB employees related to welding systems. To solve small programming issues I have found that Internet, and especially Google is the best way to go.

1.4.2 Analysis and design

The supervisor and I have decided that the analysis part of the project should cover three different ways to send the information. The analysis phase will be much shorter than the implementation phase due to time limitations. There are three major problems to be solved.

Part 1 – What unit to send the information

In this part I will try to determine what unit in the system that is best equipped to send the system information. The $W8_2$, WeldPoint or a standalone application.

Part 2 – How to send the information

The information could be sent via email or be uploaded to a FTP server or to a database server. It is important that the information does not get blocked by a company firewall. Outgoing ports except the usual ones might be blocked. It is probably impossible to provide a solution that penetrates all different firewall configurations.

Part 3 - How to store the information

This part will try to answer how the system information should be stored on the receiving side. It could be stored in a database or as files.

1.4.3 Implementation

When the analysis phase is done the implementation phase will begin. The suggested solution will be implemented in this order:

- Implement the functionality to send the information from the client (customer) side. In this stage of the project the information to be sent will just be empty files.
- Implement the functionality to receive the information at the server side.

This will be the hardest part of the implementation and when both sides can communicate with each other the graphical user interfaces will be implemented. It is best to get the communication working first before spending a lot of time finishing the graphical user interfaces.

1.4.4 Evaluation

The implemented solutions will be evaluated with a series of tests to determine that all goals and requirements are met. The implemented parts of the system will first be tested individually and then the whole system will be tested.

1.4.5 Structure

This thesis aims to give the reader a structured view of the work performed in this project and why and how different choices were made. This thesis contains different figures and diagrams in order to better explain some scenarios for the reader. In some cases the textual description might be enough to understand the context, but in most cases I prefer both a textual and a graphical description.

This thesis primarily targets persons with basic knowledge of programming, databases and computer networks.

A short summary of the structure of this thesis:

Chapter 2 - Customer's system overview

It is important for the reader to understand how the customer's welding system work in order to understand the discussion and conclusions made later on in the thesis. This chapter explains how two different customer's welding systems work. The chapter also contains the goals and requirements of this thesis.

Chapter 3 – Analysis

This chapter describes the parts in section *1.4.2 Analysis and design.* A number of different solutions are proposed followed by a discussion explaining the most suitable solution.

Chapter 4 – Web services

Explains web services and how they work. Web services are an essential part of this thesis and needs a thorough explanation.

Chapter 5 – Design and implementation

This chapter describes the design and implementation of the solution suggested in chapter *3 Analysis*. The tools being used during the implementation are also explained.

Chapter 6 – Testing

This chapter describes a number of tests that were designed to test the goals and requirements in section *2.3.1 Goals* and *2.3.2 Requirements.*

Chapter 7 - Discussion and conclusions

This chapter contains a discussion, conclusions, future work and experiences that explain the experiences I have gained during this project thesis.

2 Customer's system overview

In order to give the reader a better understanding of the decisions taken in this thesis it is necessary to understand the customer's system. The goals and requirements of this project thesis are also explained in this chapter.

2.1 The customer's welding system

Although this project will only touch a small part of the customer's entire welding system it can still be necessary to explain the whole welding system to better understand discussions later on in the report.

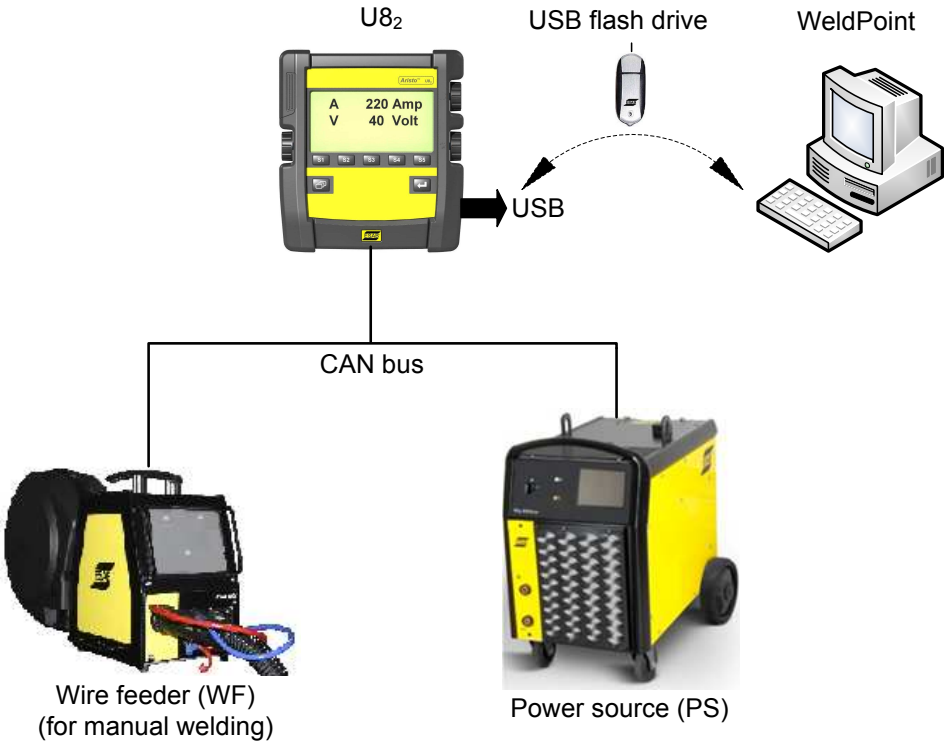

Figure 1. System overview of the customer's welding system for manual welding.

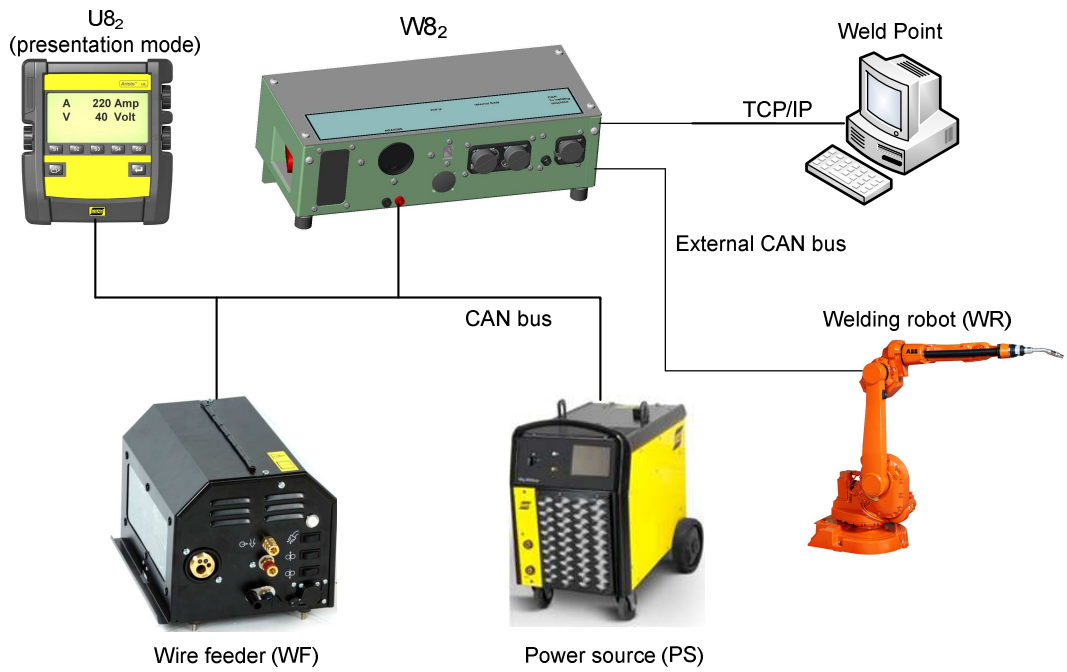

Figure 2. System overview of the customer's welding system for mechanized welding (using a welding robot).

The following text describes the units located in both *Figure 1* and *Figure 2*.

U8$_2$: This is the control unit of the system. It allows a user to set parameter values for the wire feeder and the power source. The U8$_2$ also displays the current values being used during welding. The U8$_2$ sets and retrieves the information via the CAN bus. The U8$_2$ is used by the welder (see *Figure 1*, upper middle) or the robot operator (see *Figure 2*, upper left) to change parameters for the wire feeder or the power source.

W8$_2$: When a robot does the welding (see *Figure 2*) it is called mechanized welding. If mechanized welding is used, the W8$_2$ is necessary because it communicates with the welding robot via the external CAN-bus. The W8$_2$ also communicates with WeldPoint via Ethernet.

Wire feeder: This is the unit that feeds the wire when welding. There can be different feeders for manual and mechanized welding. For example with the U8$_2$ the user can change wire feed speed.

Power source: This is the power supply. For example with the U8$_2$ the user can change current via the CAN-bus.

Welding robot: Does the automated welding. The welding robot communicates with the W8$_2$ via an external CAN bus.

WeldPoint: This is a PC-based control and log software. With WeldPoint a user can handle weld datasets and configuration settings for the system. Quality logs, error logs and production statistics can also be viewed. It is these files, mainly the log files, that are important for this thesis project. WeldPoint communicates with the W8$_2$ via an Ethernet. WeldPoint is a Windows based application written in Microsoft Visual C#.NET.

If manual welding is used like in *Figure 1* the W8$_2$ is not used. Since the W8$_2$ is the only unit in the system that can communicate with WeldPoint this means that WeldPoint cannot be used in the manual welding case. However, since the U8$_2$ has a USB connector, files can be transferred from the U8$_2$ to a PC with a USB flash drive and then viewed in WeldPoint.

2.2 The customer's internal network

It is common that ESAB's customers have several installations of the system described in *Figure 2*. Only one WeldPoint application is necessary because it can connect to 10 $W8_2$ units at the same time.

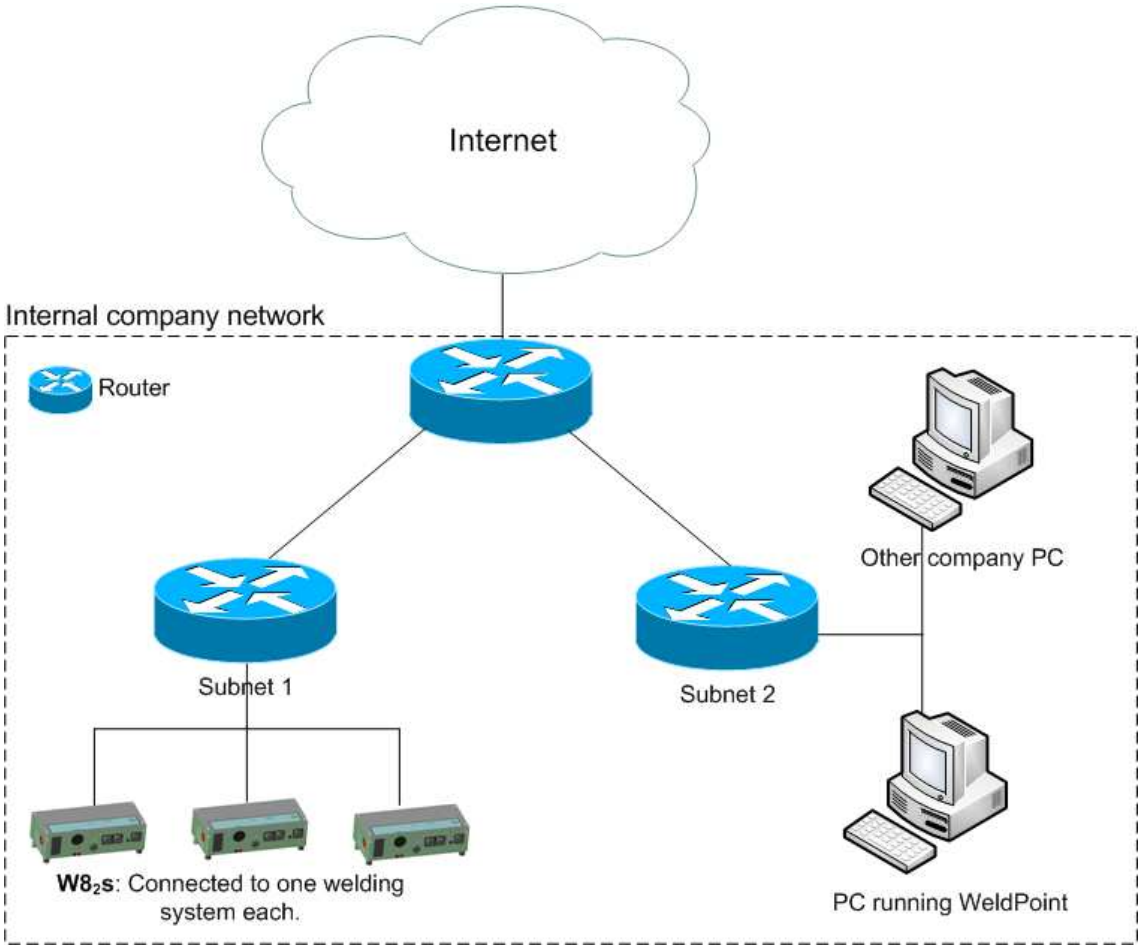

Figure 3. A typical internal network of one of ESAB's customers.

The $W8_2$s (one for each welding system) might be in a separate networks as in *Figure 3* or they might be in the same. It depends on the particular company's network configuration. The PC running WeldPoint can connect to the $W8_2$s via the network. It is important to note that the welding systems do not require Internet connectivity to work. However, if ESAB chooses to implement the solution given in this thesis the unit sending the system information will require an Internet connection. As explained in section *1.4.2 Analysis and design* the unit sending the system information will either be the $W8_2$, WeldPoint or a standalone application.

2.3 Requirement specification

This describes the overall goals and requirements of the project. Priority describes the importance of a goal/requirement. Goals/requirements with priority high will be met first.

2.3.1 Goals

The following goals have been set by me and the supervisor at ESAB.

Goal description	Priority
An analysis comparing three different ways to send the system information. The type of equipment being used, software versions, conditions that the problem occurred in, and CAN logs are examples of what the system information can describe.	High
Implement functionality to send the system information. This functionality could be implemented in WeldPoint, the $W8_2$ or a standalone application (the analysis phase will determine which one).	High
Determine how the system information will be sent over the Internet. Sent by email, upload to a FTP server or to a database server.	High
Implement functionality to receive and view the system information. The analysis phase will determine how this will be implemented.	High
Determine which log files and other information to be sent.	Medium
Implement functionality to send files from a portable USB flash drive (see *Figure 1*) using the implemented solution.	Medium

2.3.2 Requirements

The following requirements have been set by the supervisor at ESAB.

Requirement description	Priority
The information sent over the internet should not be stopped by company firewalls.	High
The information should not be sent in clear text.	High
The user applications (at both ends) should be able to run on Windows XP and Windows Vista platforms.	High
The employees at ESAB should be able to view the information sent by the customer.	High

3 Analysis

This chapter will analyse the different parts in section *1.4.2 Analysis and design*. A short description will be given about how the solution with this particular method would be done. In the end the different solutions will be discussed and a solution will be chosen.

3.1 What unit should send the information?

This section discusses whether the $W8_2$, WeldPoint or a standalone application is the most appropriate way to send the system information.

3.1.1 $W8_2$

The $W8_2$ is an embedded system with the following specification:

- NXP LPC2468 microcontroller.
- 8 MB external SDRAM.
- 4 MB external flash memory.
- 512 KB external SRAM.

The microcontroller has, among other things, a built in: 10/100-Mbps Ethernet interface, a USB 2.0 full speed external data transfer, and two CAN channels [3]. When the $W8_2$ boots, the application code in the flash memory is loaded in to the external SDRAM and run. The maximum size for application code is therefore 4 MB (the size of the flash memory). In the current version of the $W8_2$ the flash memory is about 50% full. This means that if the functionality to send the system information would be implemented in the $W8_2$ the code can be a maximum of 2 MB. Even if the free memory would be enough to implement the solution ESAB still wants a certain amount of free space for additional updates and/or addons.

Another performance issue is the CPU. When I asked the developers at ESAB the general idea is that they want to minimize the workload on the CPU. An overloaded CPU could effect the welding process.

3.1.2 WeldPoint

As stated in section *2.1 The customer's welding system*, WeldPoint is a PC-based control and log software written in Microsoft Visual C# and Microsoft .NET Framework 2.0.

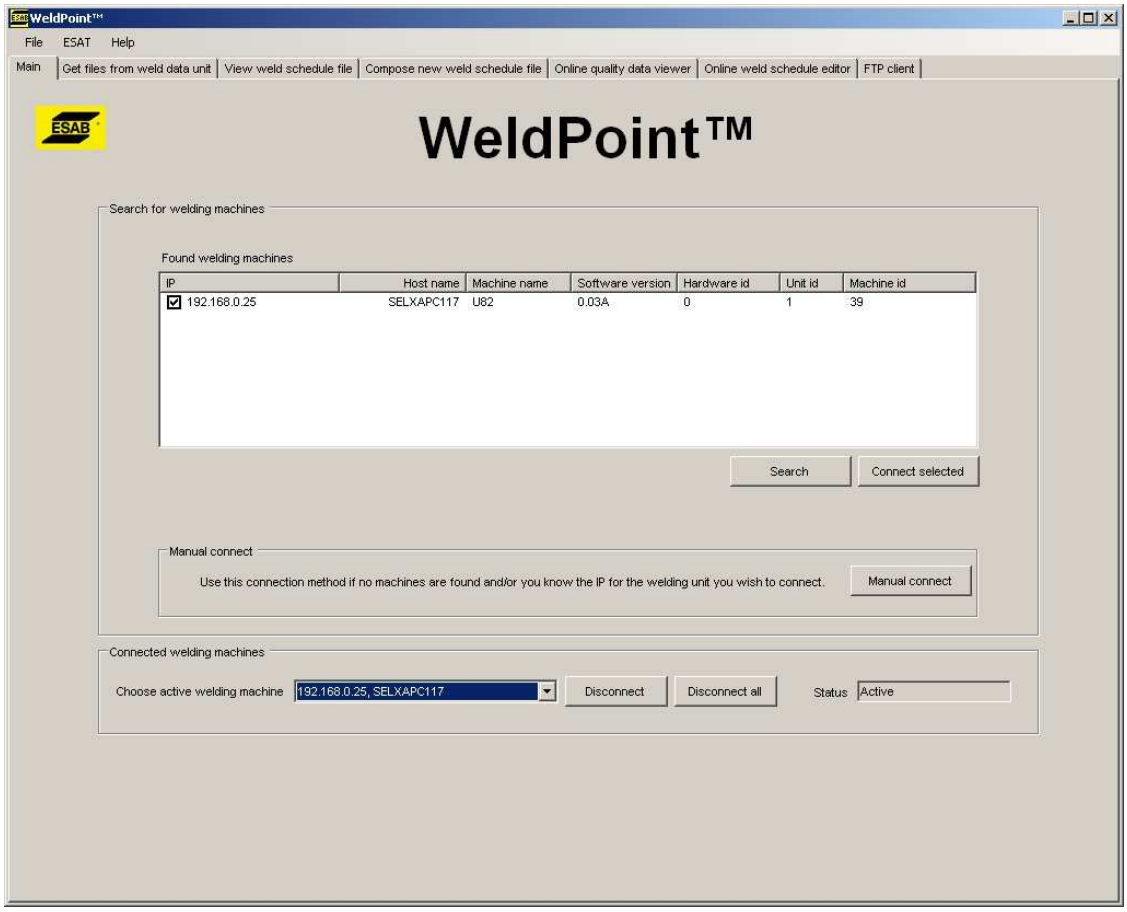

Figure 4. WeldPoint Main tab.

When the user starts WeldPoint she is presented with a screen showing the different $W8_2$s (see *Figure 4*, middle) on the network. If the user connects to one of these she can perform different functions using the tabs (upper left to right).

A new tab would be the best way to integrate functionality to send system information via WeldPoint. It would be the most user-friendly way.

C# and .NET 2.0 libraries offer functionality to implement all three solutions proposed in section *1.4.2 Analysis and design* [7].

3.1.3 Standalone application

The system information could be sent using a standalone application. This application could be a Windows application or a web-based application. For example, a user could log in to a webpage to send the system information.

3.1.4 Discussion

Having the $W8_2$ to send the system information is not a good idea because of memory and performance issues. All the issues with using the $W8_2$ to send the information can be solved by using a PC application instead. Two options remain.

Advantages of using WeldPoint to send the system information:

- Built-in functionality to communicate with the $W8_2$ (to get the system information).
- 2-in-1 application. Confusing for the user to use two different applications.

Advantages of using standalone application to send the system information:

- This could be platform independent if it is implemented as a web application or in a platform independent language like Java [8].

Except the one advantage with the standalone application there is really no reason to implement the solution as a standalone application. Also, since WeldPoint and the standalone application can be run on the same computer, the platform independency would not matter since WeldPoint can only be run on Windows XP and Windows Vista operating systems. Another disadvantage with using a standalone application would be to have to implement the functionality to communicate with the $W8_2$; this would be a very time-consuming task. The best solution in my opinion is to implement the functionality to send system information under a tab in WeldPoint and therefore the implementation programming language will be Microsoft Visual C# and Microsoft .NET Framework 2.0.

3.2 How to send the system information

This section will discuss different ways to send the system information from WeldPoint to ESAB.

3.2.1 Email

The system information from the customer could be sent to ESAB employee via email. The customer simply clicks a button and the files are downloaded from the $W8_2$ to the PC running WeldPoint and then emailed to ESAB. The files are sent along with the email using MIME. Multipurpose Internet Mail Extension (MIME) is a supplementary protocol that allows non-ASCII data to be sent through email [9]. The plain-text information such as: type of equipment and software version could be sent in plain-text. The email could be sent in two ways:

- Using the PC's default email application and there for using the customers email configuration.
- Using an external SMTP server to send the email [1].

The first way to send the information would guarantee that the system information can be sent and not stopped by for example a company firewall. However, if the company uses a web-based email system or if there is no local email configuration available then an external SMTP server has to be used. If the company uses a web-based email system it is possible that outgoing traffic on port 25 (SMTP) is blocked [1]. In this case this solution would not work. For example, ESAB's local software firewall does not allow applications to communicate on port 25.

3.2.2 FTP

Instead of sending the system information via email, WeldPoint sends the files and the plain-text using FTP (File Transfer Protocol). FTP is the standard mechanism provided by TCP/IP for copying a file from one host to another. FTP uses port 20 and port 21, which means that the company firewall must allow traffic on these ports [9]. It is possible that the FTP ports are blocked by the company firewall.

3.2.3 Database via a web service

WeldPoint could be made to communicate with a database located at ESAB. This solution has the same issues regarding port blocking by firewalls as the other two solutions above. However, all this can be avoided using web services.

Web services

A web service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network" [11]. A web service provides a service

interface enabling clients to interact with servers. External data representation and marshalling of messages exchanged between clients and web services is done in XML[13]. SOAP (Simple Object Access Protocol) specifies the rules for using XML to package messages [13]. For example an application written in Java can communicate with a web service written in C# or the other way around. SOAP is also used to encapsulate these messages and transmit them over HTTP or another protocol, for example, TCP or SMTP [10]. Because all of the data is transferred using HTTP, applications can easily communicate across firewalls and other network obstacles [6].

Security of web services could be achieved by using WS-Security. This protocol contains specifications on how integrity and confidentiality can be enforced on web services messaging [13].

Web services are more thoroughly explained in chapter *4 Web services*.

Web service communication with WeldPoint

WeldPoint (see *Figure 5*, left) could communicate with a web service (bottom right). This web service could then communicate with a database located in ESAB's internal network (right). The web service stores the system information sent from WeldPoint in the database.

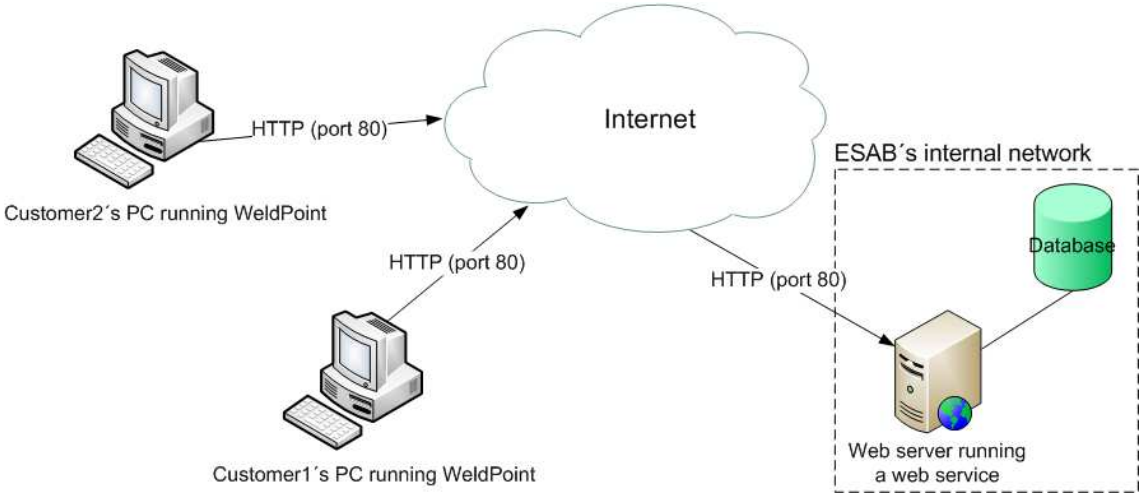

Figure 5. A web service example. The request will look just like an ordinary web page request to the firewall (except there is a SOAP message encapsulated by the HTTP packet).

The terms web server and web services should not be mixed. A web server provides a basic HTTP service, whereas a web service provides a service based on the operations defined in its interface.

3.2.4 Discussion

System information arrival is crucial. If the computer running WeldPoint has Internet connectivity, port 80 is most likely to be accepted as outgoing traffic by a firewall compared to the other two solutions. This reason alone is enough to choose the web service solution. I can not see any drawbacks with using the web service example except that a web server is needed to run the web service. However, the other alternatives also require some kind of servers.

Although the web service could be implemented in for example Java, I choose to implement it in C#. There are three reasons for this:

- It is easier to use the same programming language for the WeldPoint client implementation, the web service and the application used by developers at ESAB to view the system information.

- Visual Studio 2005 has built-in support for creating web services projects.
- The developers at ESAB have more knowledge in C# than Java.

3.3 How to store the information

The web service will receive the system information and log files from many different WeldPoint users. In order to handle this information in a structured way a database is needed. When the web service receives the system information it stores the type of equipment being used, software versions etc. in the database in specific fields and the log files as binary data.

This chapter came to the conclusion that a web service was the best way to receive the system information. *Chapter 4* will describe web services more thoroughly to give the reader a better understanding of web services. This is important to better understand the design and implementation chapter.

4 Web services

This section will describe the most important parts of a web service. It is not necessary to know exactly how web services work to implement the solution, because there are many tools available that create the XML (SOAP messages) needed to communicate with a web service. Visual Studio, for example, has built-in functionality to create and connect to web services. However, I choose to study web services a bit more thoroughly than needed because I am very interested in distributed network programming and especially web services because of its ability to communicate across firewalls and other network obstacles.

4.1 History

Web services had its beginnings in mid to late 2000 with the introduction of the first version of XML messaging – SOAP, WSDL 1.1, and an initial version of UDDI as a service registry [13]. These specifications will be explained in following section.

4.2 The web service protocol stack

The web service protocol stack is modular. This means that new modules can be added to the stack to satisfy additional requirements. Since the protocol stack is modular a developer can choose which modules to use depending on the application he or she is developing. For example, a developer might choose not to implement all security specifications within the security block that are offered by the web services protocol stack.

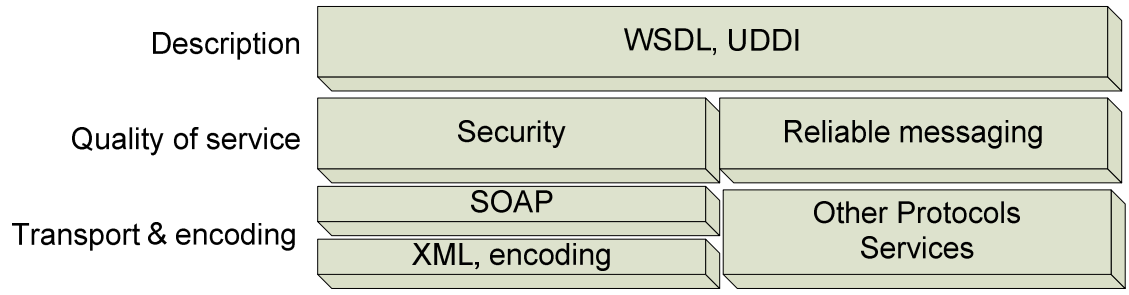

Figure 6. The Web service protocol stack [14]. This is a simplified figure. There are additional modules, but these here are the ones described in this section.

4.2.1 Transport and encoding

This section will briefly describe the first (bottom) layer in the web service protocol stack. It contains SOAP, XML encoding and other protocols services.

XML

XML (eXtensible Markup Language) is a markup language that was defined by W3C for general use on the web [13]. XML's purpose is to aid information systems in sharing structured data, especially via the Internet, to encode documents, and to serialize data [15]. Serialization is the process of converting an object into a sequence of bits so that it can be stored in a file, a memory buffer or transmitted across a network (like the Internet) [16].

SOAP

SOAP (Simple Object Access Protocol) is the fundamental messaging framework for web services. It is a lightweight, platform independent, XML-based messaging and RPC protocol (Remote Procedure Call)[14]. RPC is used as an interprocess communication mechanism used to start some process on remotely located processors [14]. You could say that SOAP is for web services what HTTP is for the web.

SOAP provides four main capabilities [13].:

- A standardized message structure based on the XML Infoset. An XML Infoset describes an abstract data model of an XML document [27].
- A processing model that describes how a service should process the messages.
- A mechanism to bind SOAP messages to different network protocols.
- A way to attach non-XML encoded information to SOAP messages.

4.2.1.1.1 SOAP message

A SOAP message is carried in an "envelope". Inside the envelope there is an optional header and a body. Message headers can be used for establishing the necessary context for a service for keeping log or audit of operations. The message body contains an XML document for a particular web service [10].

Code boxes 1, 2 and *3* contain a simple example of SOAP message communication. A client calls a method in *Code Box 1*, and it sends the SOAP message in *Code box 2*. *The SOAP-ENV:Body* in *Code box 2* contains the name of the web service, method name and the argument to the method. When the server receives the request it runs the method and constructs the SOAP message in *Code box 3*, containing the increased number and sends it. This example only sends a simple *int* to the server but SOAP defines encoding styles which will allow for the generation of an XML representation for almost any type of application data [14].

```
int doubleAnInteger(int numberToDouble);
```

Code box 1. Method to be called.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1: increaseANumber=""
                      xmlns:ns1="urn:MyWebService">
      <param1 xsi:type="xsd:int">1</param1>
    </ns1: increaseANumber></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Code box 2. SOAP request message.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1: increaseANumberResponse=""
                      xmlns:ns1="urn: MyWebService"
                      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:int">2</return>
    </ns1: increaseANumberResponse></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Code box 3. SOAP response message.

4.2.2 Quality of service

This section will briefly describe the most important parts of the second layer (middle) in the web service stack. This layer consists of security and reliable messaging protocols/specifications.

Security

The security mechanism used in web services can differ a lot. You can use simple authentication methods like HTTP Basic Authentication or more advanced mechanisms with support for message encryption and nonrepudiation [14]. A more advanced mechanism is WS-Security (Web Service Security) [28]. WS-Security describes how to attach signatures and encryption header to SOAP messages [28]. This means that it is only the SOAP message that is encrypted instead of the whole HTTP message, if SSL/TLS where to be used [29]. With WS-Security you can also use X.509 certificates or Kerberos tickets to authenticate and encrypt communications [28].

Reliable messaging

SOAP messages sent over the internet with HTTP are not guaranteed to arrive to its destination. Although TCP is reliable SOAP messages can still get lost [13]. TCP only ensures that packets in the TCP/IP stack are delivered, but messages might not reach the TCP stack on the sending side. Also, on the receiving side they might not reach the application layer where the application resides [14]. This is why SOAP needs reliable message delivery. One specification to ensure reliable messaging is WS-RM (Web Services Reliable Messaging).

4.2.3 Description

This section will describe how web services can be described and how they can be discovered by clients.

WSDL

WSDL 1.1 (Web Services Description Language) is an XML-based language and is the de facto standard for describing web services. WSDL describes what services a web service provides, where the web service is located and how to communicate with it *(see Code box 4)*. WSDL is the second most used web service specification after SOAP [13].

```
<definitions>
  <types>
    ...
  </types>

  <message name="..">
    ...
  </message>

  <portType name="..">
    ...
  </portType>

  <binding name="..">
    ...
  </binding>

  <service name="..">
    ...
  </service>
</definitions>
```

Code box 4. WSDL 1.1 XML syntactic structure [13].

<Types>: This element is used to declare data structures that are referred to later in the WSDL file.

<Message>: Describes the messages that the web service is exchanging.

<PortType>: Defines the web service, the operations that can be performed and the messages that are needed to perform the operation. Each operation can be compared to a common method call. Using the previous example presented in *Code boxes 1*, *2* and *3* the <*portType*> would look like this:

```
<portType name="MyWebService">
  <operation name="increaseANumber">
    <input message="tns:increaseANumberSoapIn" />
    <output message="tns:increaseANumberSoapOut" />
  </operation>
</portType>
```

Code box 5. Example of <portType> and <operation>.

<Binding>: The <*Binding*> element tells the client how to format the messages to be able to interact with the web service.

<Service>: The <*Service*> element is the final part of the WSDL file. Its children elements <*port*> describe where to find a service. The <*port*> describes what <*portType*> is offered via a given <*Binding*>. The <*Port*> also contains the address location at which the binding is offered [13].

UDDI

UDDI (Universal Description, Discovery and Integration) is directory service to make services available to clients. For example, a travel agent can publish its services using a WSDL file and thereby making it available to clients, or other travel agents wishing to communicate with the travel agent. UDDI will not be used in this thesis but it is an important part of web services.

Chapter 5 will among other things describe the design and implementation of the web service, using the web service theory explained in this chapter.

5 Design and implementation

This chapter will describe the design and implementation of the WeldPoint GUI, the web service, the database, and the GUI for the ESAB employees. From here on I will use the following names and abbreviations:

- The WeldPoint GUI will be called WeldPoint Remote Support (WRS).
- The web service will be called WeldPoint Web Service (WWS).
- The GUI for the developers at ESAB will be called WeldPoint Remote Support Center (WRSC).

This chapter is divided into four parts. First the tools used in this thesis will be explained. Then the design and implementation of all four parts (WRS, WWS, the database and WRSC) will be described. The design and implementation part will describe the applications in the same order the system information is sent from a customer (WRS), until it arrives at ESAB and the employee can view it in WRSC.

The information flow can be seen in *Figure 7* below, starting at WRS to the left. A customer running WRS (left) sends the system information via HTTP over the Internet to ESAB's web server running WWS (bottom right). WWS then stores the system information to the database (middle right). The ESAB employees can then view the system information with WRSC (upper right). WWS, the database and WRSC are all located within ESAB's internal network (right). The solution in *Figure 7* is very modular. For example, the web server running WWS is IIS but it could easily be replaced with another web server such as Apache.

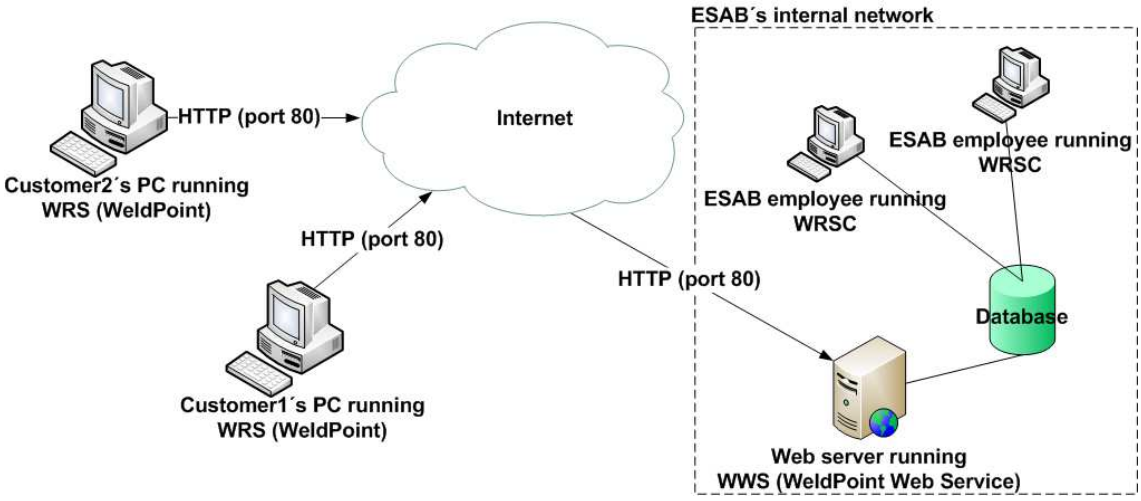

Figure 7. WRS to WRSC communication example.

WWS will partly be described in both WRS and the database since it communicates with both of them.

5.1 Tools

This section will describe what tools I chose to use during this thesis project. ESAB had no demands on what tools they wanted me to use. Each tool is described and a short explanation is given, explaining why I chose the tool.

5.1.1 C# programming language

C# is an elegant and type-safe object-oriented language that gives you the opportunity to build a variety of secure and robust applications that run on the .NET framework [5]. C# can be used to create traditional Windows client applications, XML web services, distributed components, client-server applications, database applications and much more [4]. The reason why I chose C# as implementation language was because WeldPoint was written in C# and because ESAB's developers have more knowledge of C# than of for example Java.

5.1.2 Microsoft .NET Framework

The Microsoft .NET technology is a programming platform for developing applications for Microsoft Windows workstations and servers. The core of the Microsoft .NET technology is the Common Language Runtime (CLR) environment [6]. This environment enables the developer to create programs using a multitude of programming languages; C# is one of them. These applications can then be run on any platform that supports the CLR. The idea of the CLR is to provide a middle layer of Application Program Interfaces (APIs) that operate between the low-level Windows Win32 API functions and the application program code. By providing a common middle layer, Microsoft has given a large number of application languages access to core Windows technologies (such as network support) [6]. The latest version of .NET is 3.5. I have used .NET 2.0 for all applications [30]. The reason for this is that WeldPoint is written in .NET 2.0 and ESAB did not want to upgrade the existing WeldPoint project to .NET 3.5.

5.1.3 Microsoft Visual Studio 2005

Visual Studio is an Integrated Development Environment (IDE). One can develop various applications using Visual Studio, such as Windows forms applications, web sites, web applications and web services. For example it is very easy to create a simple Windows application without even writing a single line of code. All code is generated in the background when the developer drags-and-drops components to create a Windows application. Although to create more advanced applications like in this thesis project one has to write a lot of code on your own.

In 2007 Visual Studio 2008 was released. The reason why I am not using this version is because ESAB have not yet upgraded to the 2008 version.

5.1.4 Microsoft SQL Server 2005 Express Edition

In the beginning of this project I decided to use MySQL as database server, simply because it was free and I had used it a lot in different courses during my education. When installing Microsoft Visual Studio, by default SQL Server 2005 Express Edition is also installed. Since it was already installed I tried it out and read about it. I also installed the Microsoft SQL Server Management Studio Express (MSSMSE) which is a tool for configuring, managing, and administering all components within Microsoft SQL Server [31]. The tool I intended to use for managing the MySQL database is called phpMyAdmin. One of the reasons why I chose to use Microsfts SQL Server was that MSSME in my opinion was better than phpMyAdmin, also SQL Server 2005 has some good features, which is described in section *5.4.2 Implementation*. SQL Server 2005 Express Edition and MySQL are both freeware.

5.1.5 Microsoft Internet Information Services (IIS)

The IIS offers a range of services, such as HTTP [17]. IIS is the second most used web server in the world after Apache [18]. IIS comes bundled with Windows XP and is free to use. I chose to use IIS over Apache because it is easier to deploy web services created in Visual Studio to IIS. Also, ESAB have web servers running IIS, so choosing IIS as web server will help ESAB to deploy the solution presented in this thesis project.

5.1.6 Web Services Enhancements (WSE) 3.0

WSE is a developer tool to incorporate security into web services. WSE can be used to implement message-level security [21]. WSE uses WS-Security and other WS-* specifications (see section *5.2.3 Implementation*). WSE enables you to use several different security scenarios [21]. Message Transmission Optimization Mechanism (MTOM), is also a part of WSE 3.0. MTOM is a W3C recommendation for sending large amounts of binary data. MTOM allows you to send binary data inside a SOAP message [21].

5.1.7 Virtual Box

Virtual Box is a x86 hardware virtualizer developed by SUN [19]. This software allows the developer to run extra instances of for example Windows XP, Windows Vista or Windows 2003 Server on one's own computer. The reason why I used several installations of operating systems was to try out the web service (before testing it live), sniff network traffic and try the different GUIs on different operating systems.

5.1.8 Wireshark

Wireshark is a network protocol analyzer [20]. I used this tool to better understand how web services work and to make sure that these worked the way I intended to.

5.1.9 ZoneAlarm

Zonealarm is a highly configurable software firewall for Windows. Zonealarm allows one to open ports for both incoming and outgoing traffic[25]. This is important when testing web services ability to communicate across firewalls.

5.2 WeldPoint Remote Support (WRS)

As already stated in section *3.1 What unit should send the information?* WRS will be implemented as a tab in WeldPoint. WRS is the only GUI in this solution that the customer will use. See *Figure 7* on page *17* to get a better overview and understanding of what this section describes.

5.2.1 What information should be sent?

First off I needed to determine what system and contact information ESAB want from the customer and how to get it. After discussing and consulting ESAB's developers I composed the following list of general information and system information that is relevant for ESAB:

A case is a collection of case-general information, case-system information and case-files.

Case-general information: This information is supplied by the customer.

- *Company*, name of the company that is sending the system information.
- *Name,* name of the employee sending the information.
- *Email,* employee's email address.
- *Phone,* employee's phone number.
- *Category,* in what part of the system the error occurred. Options: WeldPoint, ESAT, Panel, or other.
- *Case subject,* subject of the case.
- *Case description,* description of the circumstances in which the error occurred in.

Case-system information: This information is supplied by WeldPoint.

- *IP,* the IP of the connected $W8_2$.
- *Hostname,* the hostname of the connected $W8_2$.
- *Name,* the unit name of the connected $W8_2$, corresponds to a specific Unit ID.
- *Version,* software version.
- *Unit ID,* unit number.
- *Machine ID,* type of unit.
- *Hardware ID,* hardware version.

Case-files:

The customer should be presented with a number of different log and error files. From these files she chooses which ones to send. The option to add one's own files should also be given.

5.2.2 Design

This section describes the graphical design and the security design of WRS.

Graphical design

The best way to extract all the information from a customer is a simple form where the case-general information can be filled in by the customer and the case-system information is prefilled. The form should be easy to understand and have pop-ups describing each form field when the customer moves the mouse cursor over it. It should also be clear that it is optional to send case-files. When the form has been filled in the customer can choose whether to add case-files, depending on the error the customer wants to report.

Security design

WWS is the most security-critical application in the thesis project. WRS must authenticate itself before using WWS. It is not important to know what customer that wants to connect to WRS, only that it really is a customer. In fact it is impossible to know this because currently WeldPoint does not have any customer registration.

The case-information and case-file data should be encrypted. The data being sent is not extremely sensitive but ESAB does not want it sent in clear text.

5.2.3 Implementation

This section describes the graphical design of WRS and how it works. The communication between WRS and WWS is also explained.

Graphical design

I have tried to make the form look just like an ordinary web form, like for example when one registers for an email account. The case-general information and case-system information have been separated to make the form look less complicated. Sometimes the customer will not even have to view the case-system information, but this information is important for ESAB's employees and because of this the case-system information will always be sent.

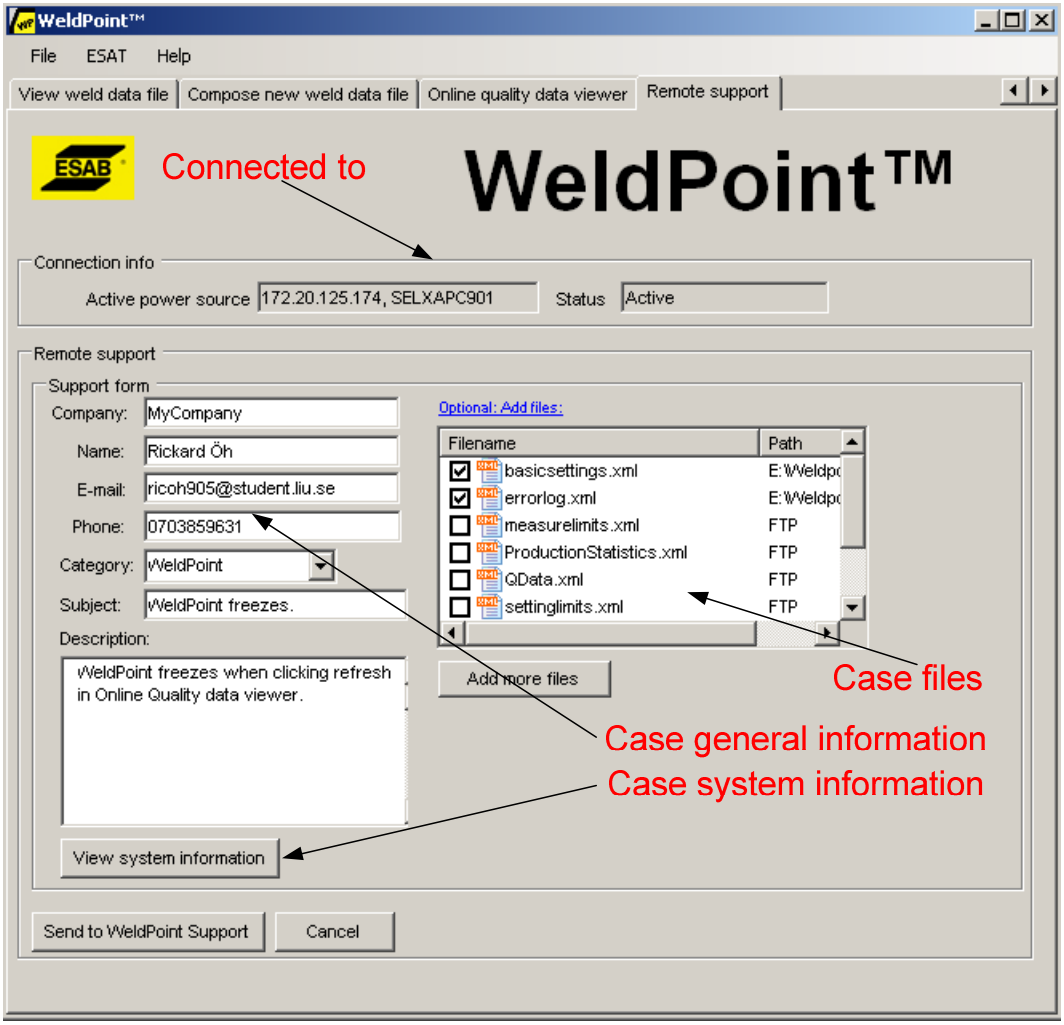

Figure 8. The WRS form. The text with arrows are descriptions.

In the figure above the *Connection info* groupbox (upper middle) shows if WeldPoint is connected to a $W8_2$, if it is not, the whole *Remote Support* groupbox (middle) is disabled and the customer cannot fill it in. The customer can fill in the contact information and the error description (left). All the fields have an input maximum and the maximum file size is 32 MB. These restrictions have been implemented to not overload WWS and the database. In this figure the customer has decided to send the files, *basicsettings.xml* and *errorlog.xml* (right).

When the customer enters the WRS tab in WeldPoint the case-files list view is always filled with nine files. The case-files list view is the list of files with checkboxes located to the right in the figure. These files are the ones located in the $W8_2$. When the checkbox next to the file is checked the file is first exported to a USB flash drive in the $W8_2$ and then transferred via FTP to the local WeldPoint computers harddrive ready to be sent to WWS when the customers clicks the button "*Send to WeldPoint Support*" (bottom left).

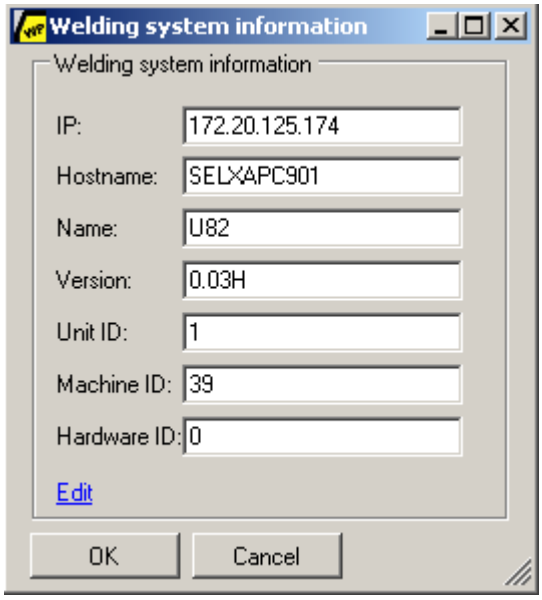

Figure 9. Pop-up window for case-system information. This is presented to the customer when he clicks the button "View system information" (see Figure 8, bottom left).

The customer can edit the case-system information but it is not recommended.

Sending the case: WeldPoint Remote Support (WRS) and WeldPoint Web Service (WWS) communication

When the customer clicks the button "*Send to WeldPoint Support*" the whole case is sent to WWS. This section will first show an overview of the communication between WRS and WWS. The case-files are not transferred in the same way as the case-general information and case-system information. A detailed description of both the case-file sending and the case-general and -system information will be given.

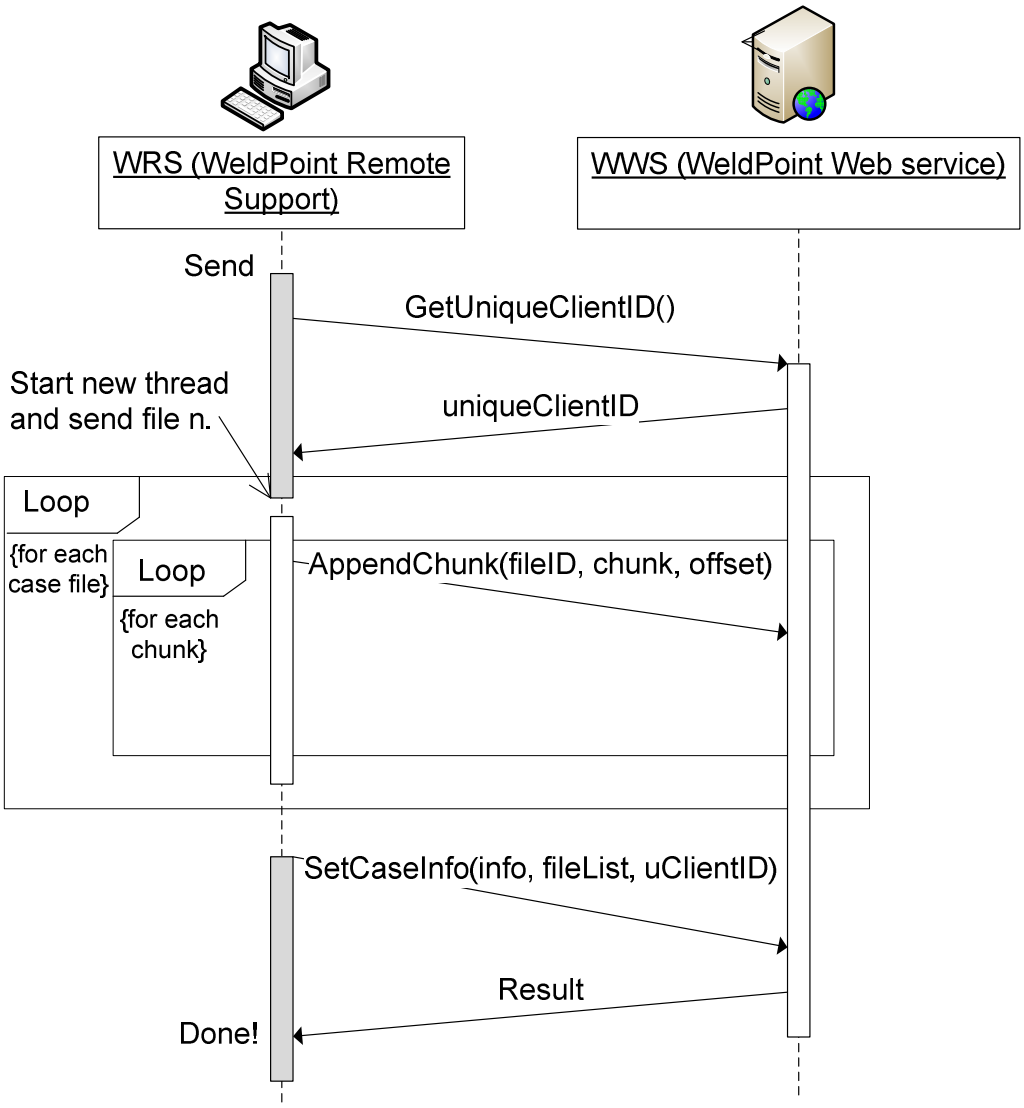

Figure 10. Messages being sent when sending a case. All these messages can be viewed in the WSDL file in Appendix A.. UDDI described in section 4.2.3 Description, is not used to discover the web service.

The gray boxes represent messages sent by the main thread. The main thread is the thread running the WeldPoint application. The white boxes are threads started at runtime.

Initialization (Two first messages in *Figure 10*)**:**

When the customer has filled in the form correctly and presses "*Send to WeldPoint Support*" (see *Figure 8*, bottom left), WRS connects to WWS and requests a unique client ID. This ID is used one way or the other in the messages shown in *Figure 10*. The *uniqueClientID* is generated by a special function in WWS ensuring that clients cannot have the same ID.

Sending case-files: (Outer and inner loop in *Figure 10*).

Each file is divided in chunks and sent to WWS. Every file to be sent is transferred by a thread created by the main thread. If the main thread were to send the file the application would freeze until the whole file had been sent. If the customer would change his or her mind and cancel the sending of the case, this would not be possible because the *Cancel* button would not respond (see *Figure 8*, bottom left). The reason for this is that the main thread is busy sending the file.

When a chunk is received by WWS (*AppendChunk* message is received) it is temporarily saved to the web server's hard drive. The reason for this is to not overload the web server's RAM. If many clients where to send files at the same time the process running WWS would soon be out of memory. The chunk size is adjusted automatically by WRS depending on the speed of the connection.

The argument *fileID* is the *uniqueClientID* + 0 + n. For example, if a client has *uniqueClientID: 342efbca96975a* then the first *fileID* would be *342efbca96975a**00*** and the next one *342efbca96975a**01***. Because of this, a case can at most contain 100 files, which is more than enough. The reason why *fileID* is needed in WWS is because it saves the files temporarily to the web server's hard drive and the *fileID* is used as the filename of the file. If two files would have the same filename the files would be corrupted since WWS could write the wrong chunk to wrong file.

The argument *chunk* is a byte array containing the chunk data for the file being sent. When the chunk is received at WWS it is appended to the file *fileID* (stored on the web server's hard drive).

The argument *offset* is the offset to write the chunk to.

When all chunks have been transferred (the inner loop has finished), the file-sending thread is terminated. If there are more files to be sent, a new thread is started by the outer loop in the figure and the next file is sent by this thread.

The files are sent using MTOM - Message Transmission Optimization Mechanism. MTOM is a W3C recommendation for sending large amounts of binary data. MTOM allows you to send binary data inside a SOAP message [21]. This means that WS-Security can be used when sending the files just like when sending the case information.

Sending case information: (*SetCaseInfo* and *Result* message in *Figure 10).*

When all files have been sent (the outer loop has finished), the main thread sends the case-general information and case-system information using the *SetCaseInfo* message.

The argument *info* is an object of a class called *CaseInfo* and is defined in WWS. The developer can add WWS as a web reference in the client project (WRS) in Visual Studio. This means that one can use classes defined in WWS in WRS! The definition of *CaseInfo* can be viewed in the WSDL file in *Appendix A*. *Info* contains both the case-general information and case-system information.

The argument *fileList* is a list of all the actual filenames that have been used during the file-sending phase (the outer loop). For example: the file *basicsettings.xml* in the case-files list view in *Figure 8* would have the *fileID: 342efbca96975a00* during the case-file sending part, but WWS needs to know the actual filename which is supplied in *fileList.* The filenames in *fileList* are stored along with the binary file in the database.

The argument *uClientID* is the same as *uniqueClientID* in the message exchange. WWS needs to know this in order to determine which files stored on the local hard drive that belongs to a particular case.

When WWS has received the *SetCaseInfo* message it returns *Result*. If WWS stored the case successfully to the database *Result* is *true*. If *Result* is *false* then an error message is presented to the customer by WRS.

Security

This section only describes the security concerning the communication between WRS and WWS. WSE 3.0 described in section *5.1.6 Web Services Enhancements (WSE) 3.0*, enables the developer to use different security scenarios to secure a web service. Two of these scenarios are Kerberos and AnonymousOverCertificate. Kerberos is an authentication protocol but this solution should mostly be used within an intranet and it requires extra functionality by the

server running WWS [21]. I choose to use AnonymousOverCertificate. This security scenario allows WRS to authenticate itself using WWS's public X.509 certificate (which contains WWS's public key) and then communicate securely [21].

WWS's public X.509 certificate is shipped with WeldPoint. Every time WRS want to communicate with WWS a check is performed if the certificate is installed on the local computer. If the certificate is not installed WRS will install it to the current local Windows user's certificate store and then use it. WWS considers WRS authenticated if it has WWS's public X.509 certficate.

AnonymousOverCertificate works like this:

1. WRS encrypts the message to be sent with WWS's public X.509 certificate (public key). A temporary symmetric key is supplied by the client when sending the message.
2. The message is sent.
3. When the message is received by WWS, it is decrypted with WWS's private key.
4. The response is encrypted with the symmetric key supplied by WRS in step 1.
5. The client receives the response from the server and decrypts it with the temporary symmetric key.

Security scenarios are implemented in WRS and WWS as policy files. The policy files describes for example what scenario to use and where the local certificate is stored. WRS's policy file is not identical to WWS's.

5.3 Database

I chose to use Microsoft SQL Server 2005 Express Edition as database server and Microsoft SQL Server Management Studio Express to maintain and create the database.

5.3.1 Tables

The database contains three tables, *Case*, *File* and *User* and are all normalized in BCNF. The ER diagram for the database can be viewed in *Appendix B*.

The *Case* table contains all the case-general information and case-system information and some additional attributes, such as *viewed* and *solution*, which are used by WRSC. Each case has a foreign key relationship with the user it is assigned to; a case can only be assigned to one user. This is explained further in section *5.4 WeldPoint Remote Support Center (WRSC)*.

The *File* table contains all the case-files and information about the files. The actual file data is stored as a *varbinary(max)* which is an attribute called *binaryFile* that contains binary data. The maximum size of *varbinary(max)* is 2 GB but WRS prevents the user from sending files larger than 32 MB so this is not an issue. I have chosen not to set a limit smaller than 2 GB because ESAB might want to change the upper limit. Each file has a foreign key relationship with the case it belongs to; a single file can only belong to one case.

The *User* table contains all the WRSC users. This table is only used by WRSC and has nothing to do with WRS.

All attributes have a maximum value.

5.3.2 Stored procedures

Instead of sending SQL commands to the database server one can call procedures located on the server that perform a specific task on a table.

```
PROCEDURE [dbo].[ChangeStatus]
           @cID int,
           @status int
AS
BEGIN
           SET NOCOUNT ON;
           IF (@status = '1')
                      BEGIN

                      UPDATE Case SET status = @status
                                 WHERE cID = @cID

                      END
           ELSE
                      BEGIN

                      UPDATE Case SET status = @status,
                                 username = null
                                 WHERE cID = @cID
                      END
END
```

Code box 6. Example of a simple stored procedure used by WRSC to change status of a specific case. @cID and @status are variables set by WRSC before calling the procedure.

Stored procedures have many advantages:

- Possible to set access permissions for different database users.
- Can prevent SQL injections attacks.
- Allow modular programming. For example, the stored procedure above is called from four different places in WRSC. If I used ordinary SQL statements and wanted to change the SQL command then the code would have had to be updated at four different places in the code. However, with stored procedures it will only be necessary to change the code at one place, the database!
- Reduce network traffic. Instead of sending large SQL statements, a single procedure can be called to perform the statements [22].

Almost all database communication between the database and WWR and WRSC is performed using stored procedures.

5.3.3 Storing the case: WeldPoint Web service (WWS) and database communication

This section will describe how WWS adds a case to the database. As *Figure 7* on page *17* shows both WWS and the database are residing inside ESAB's intranet. The database can only be accessed from within ESAB's intranet. All external communication with the database goes through WWS.

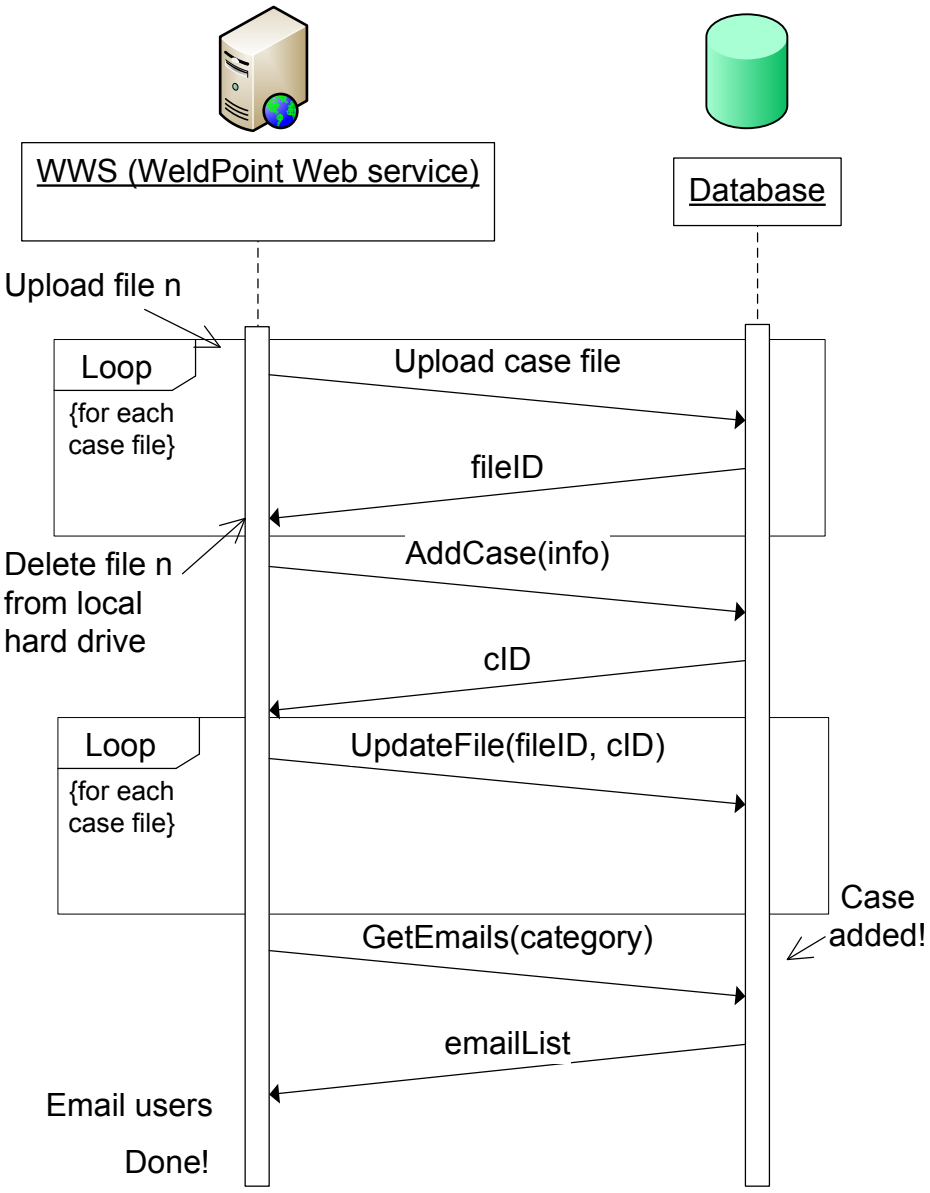

Figure 11. Communication between WWS and the database when adding a case to the database.

When WWS has received the last message in *Figure 10*, the case information is stored in WWS and the case-files are stored on WWS's local hard drive.

The case-files are then uploaded from WWS's local hard drive to table *File's* attribute *binaryFile* (first message in *Figure 11*). For each case-file that is uploaded the *fileID* (primary key) of that row is returned. When a file has been uploaded successfully to the database it is removed from WWS's local hard drive.

After all the case-files are uploaded, the stored procedure *AddCase* (the second message being sent from WWS) is called. This procedure creates a new row in table *Case* and fills it with *info*, which is the case-general information and -system information. Then the *cID* (primary key) of the new row is returned. The *cID* is then used as an argument together with *fileID* to call the stored procedure *UpdateFile* (the third message sent from WWS). This procedure updates each created *File* row's foreign key to refer to the created case. Now the case is added to the database and ready to be handled by an ESAB employee!

When the case has been added to the database, ESAB's employees need to be notified that a customer has sent a case. The WRSS users will be notified by email that a new case has been received from a customer. WRSC and the users will be explained in section *5.4 WeldPoint Remote Support Center (WRSC)*.

5.3.4 Security

Since the database is only accessible within ESAB's intranet, the security is not as important as with WWS, although some kind of authentication must exist. When WWS connects to the database it supplies a username and password to access it. The connection is not encrypted.

5.4 WeldPoint Remote Support Center (WRSC)

WRSC is the application that ESAB's employees will use to view the cases. It has been implemented as a stand alone application written in C#. WRSC will only be used within ESAB's intranet.

5.4.1 Design

ESAB had no requirements regarding WRSC but I have had several discussions with my supervisor concerning the design and implementation of WRSC. We came to the following conclusions:

- WRSC will be used by both ESAB service center employees and ESAB developers.
- A user should be able to get an overview of all the cases.
- WRSC should have user management. All users of WRSC must have user accounts.
- WRSC should be able to have several users logged in at the same time.
- A user should be able to assign cases to other users.
- When a case is assigned to a user, an email notifying the user that she has been assigned a case should be sent.
- The cases could be listed by category (WeldPoint, ESAT, Panel, Other) or by the user they are assigned to.
- A case should be marked as *unsolved*, *solved* or *assigned to a user*.
- When a case is marked as solved, a solution should be given by the WRSC user marking the case as solved.

Use-case scenario

A simple use-case, after a case has been received. Consider two WRSC users.

User 1: ESAB service center employee.

User 2: ESAB developer.

- User 1 gets a notification email from WRS saying a new case has been received.
- User 1 logs in to WRSC and views the case. The case category is WeldPoint. User 1 realizes that she can not solve the case, because she does not understand the error description or she suspects it is a bug in WeldPoint causing the error. User 1 assigns the case to User 2.
- User 2 gets an email that she has been assigned to a case. He logs in to WRSC and reads the case. User 2 solves the case, contacts the customer and marks the case as

solved. When User 2 marks the case as solved she is presented with a new window asking her to provide the solution for the case. The solution is stored in the database.

5.4.2 Implementation

This section describes the graphical design of WRSC and how it works.

Graphical design

My main goal when designing the GUI was to make it as easy to understand as possible. A user should be able to understand most of the features of the application just by looking at the graphical interface. Therefore, when designing WRSC I have had the look and feel of an email application in mind.

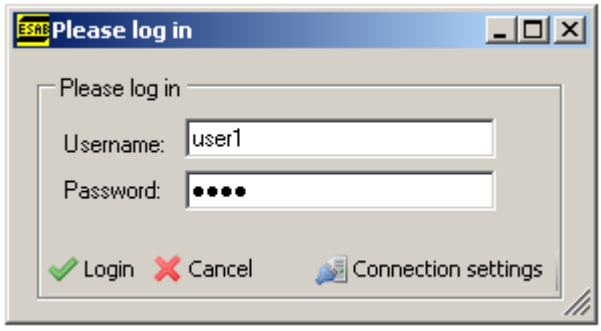

Figure 12. WSRC log- in.

When a user starts WSRC she is presented with a log-in form.

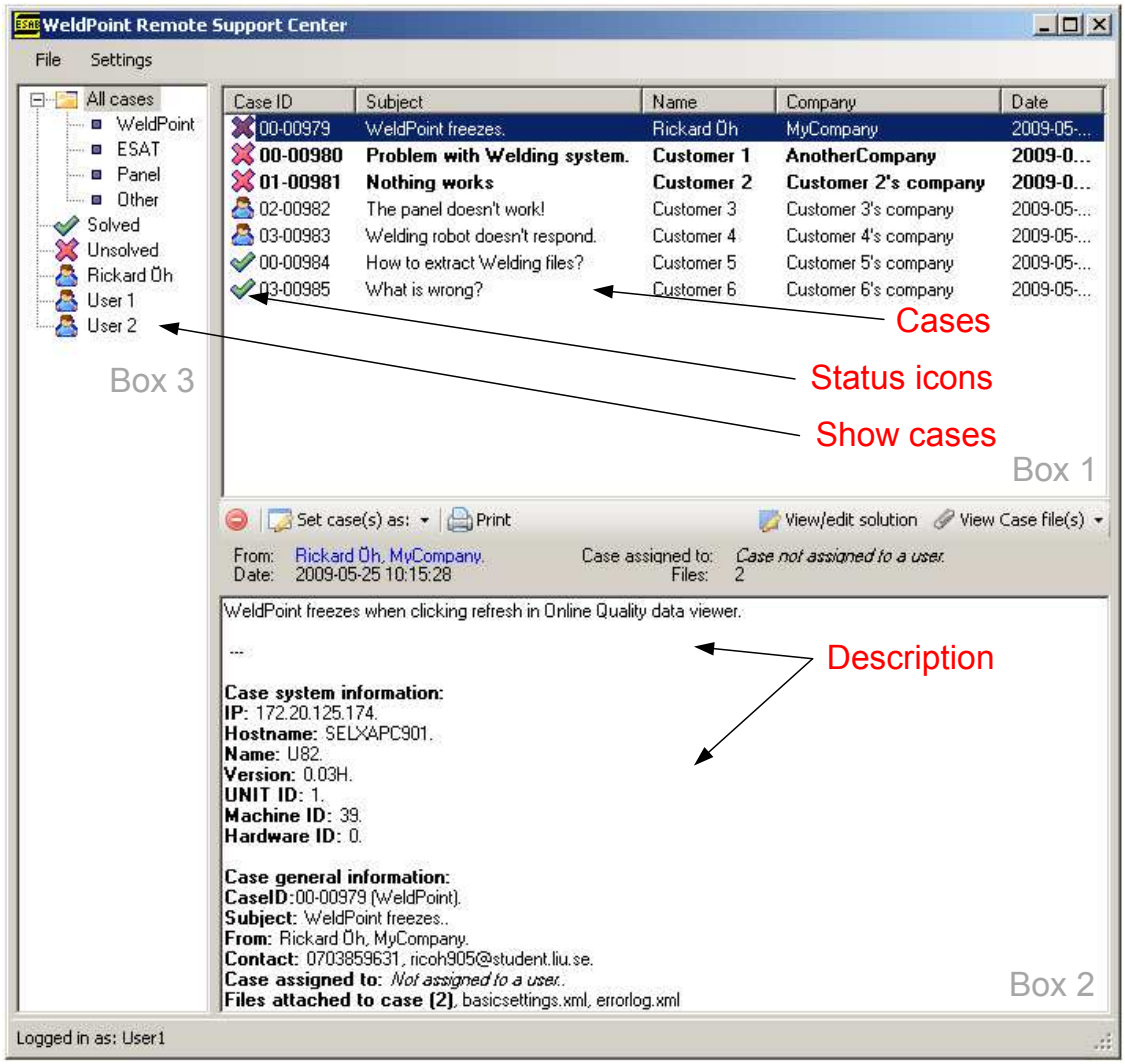

Figure 13. The WRSC application.

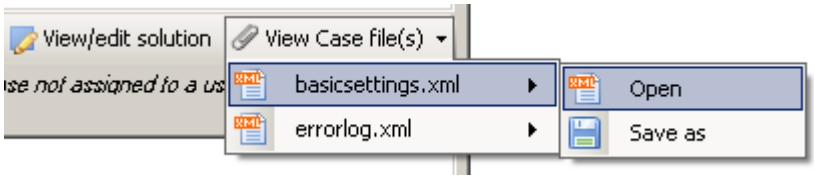

Figure 14. The View Case file(s) menu (one case must be selected).

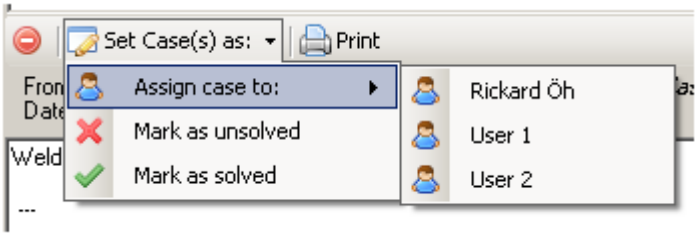

Figure 15. The Set Case(s) as: menu (at least one case must be selected).

When the user has logged in (see *Figure 12)* successfully she is presented with the main WRSC application presented in *Figure 13*. The graphical design of WSRC looks similar to an email application, but with some extra features.

The cases are presented to the user in *Box 1* (see *Figure 13*, upper right), depending on the option chosen in *Box 3* (see *Figure 13*, upper left). *Box 1* can either present all cases, cases of a specific category, all solved cases, all unsolved cases or the cases assigned to a specific user. All users in the system are presented in *Box 3* as person icons.

The status icon located next to the *case ID* indicates the current status of a case. The case can have one of the following statuses: assigned to a user, unsolved or solved. Any user can mark a case as unsolved or solved or assign it to any user.

When a case is clicked in *Box 1* the information about the case is presented in *Box 2* (see *Figure 13*, lower right). The case description, case-system information and case-general information are presented. When the case is selected the user can do a number of different things. The user can for example view the case-files by clicking the menu button *View case file(s) (*see *Figure 14*) or assign the case to a user (see *Figure 15*, middle). When a user chooses the option *Open* or *Save as*, the case-file is downloaded from the database and stored locally on the user's computer.

User management

WRSC users can be added, removed or edited in WSRC User management (see *Figure 16*) by first clicking Settings and then selecting Users from the menu that appears. When creating a user there is an option called *Choose notification*. This option contains four different categories (WeldPoint, ESAT, Panel, Other). When a new case is received all users will receive a notification by email if they are a member of one of the categories described above. This email is sent by WWS (*see Figure 11*). A user can be a member of any number of categories. Users are stored in the database table *User*, which can be viewed in *Appendix B*.

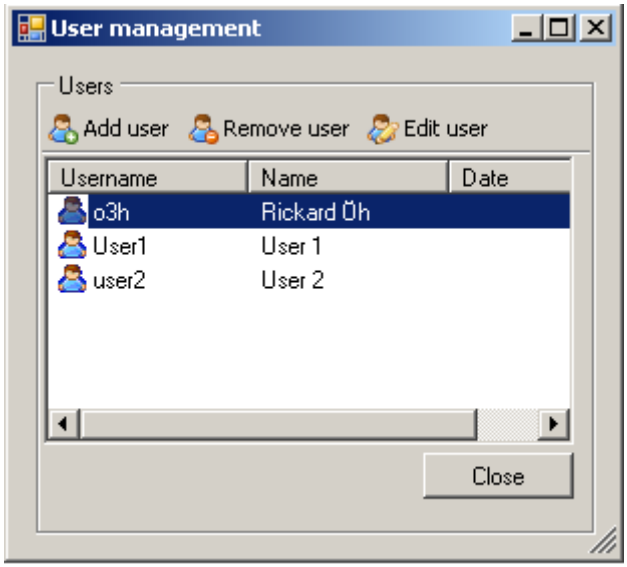

Figure 16. User management window of WRSC.

Presenting the case: WRSC (WeldPoint Remote Support Center) and database communication.

I have tried to minimize the database communication as much as possible and make WRSC run more smoothly. For example when the user logs in and all the cases are presented, WRSC only downloads the necessary information. A description is only downloaded from the database when a user clicks a specific case.

After the log-in all the necessary case data from the database table *Case* and table *User* are downloaded using stored procedures in the database and then stored locally in a *DataTable*. A *DataTable* is a class representing an in-memory cache of a database table (or a part of a

database table) [23]. The case and user information is then retrieved from the *DataTables* and presented to the users.

Multiple users

Several users should be able to use WRSC at the same time. This creates many problems that need to be solved. All users must have the same view of the application. For example, if a user assigns a case to a user, all the other WRSC applications connected to the database must be notified and updated with the new case assignment immediately. Another example is if a user deletes a case or changes its status. To solve the notification and update issue I used the *SqlDependency* class.

The *SqlDependency* object represents a query notification dependency between an application and a database, this is a new feature in SQL Server 2005 [24]. WRSC registers to receive notifications when a specific attribute or a table in the database changes.

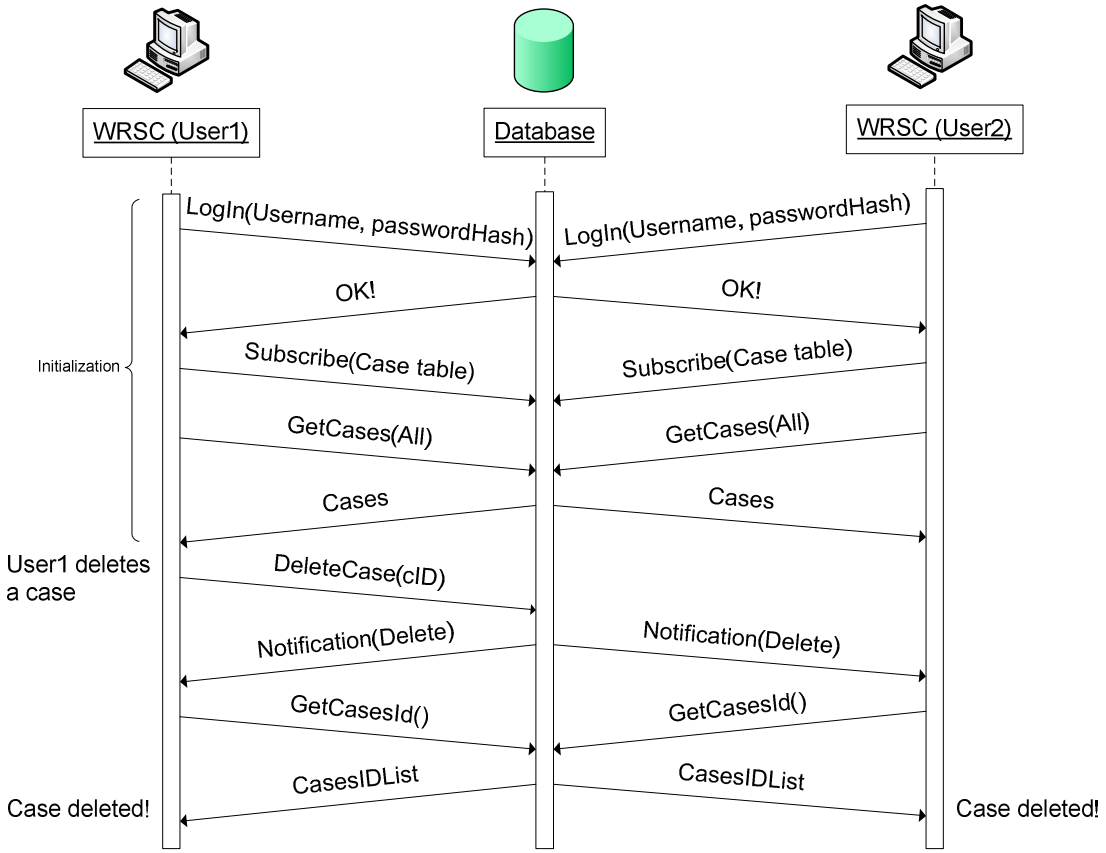

Figure 17. Simplified query notification example for when a user deletes a case.

In the initialization part in *Figure 17*, two WRSC users log-in and subscribe for the query notification of table *Case*. User 1 deletes a case using the stored procedure *DeleteCase(cID)* (middle of the left bar). The database notices that a change to table *Case* has been made and notifies all connected WRSC applications. All the WRSC applications then request a list of all *case IDs*, then this list is compared to the local *DataTable* of cases and the case is removed locally for all WRSC applications.

If a user for example assigns a case to a user the database sends the message *Notification(Update)*, since a specific case attribute *assignedToUser* has been updated from *null* to the username of the user. *Notification(Insert)* is sent if a new case has been received.

Security

Since WRSC is only intended to be used within ESAB’s intranet the security is not as important as with WWS. WRSC connects to the database the same way as WWS does. It supplies a username and a password and the connection is not encrypted.

6 Testing

This section will describe a series of tests that were performed on WRS, WWS, the database, and WRSC. Both individual and larger tests where the whole system is involved were performed. All tests will be described one by one and the result will then be presented and explained. These tests were designed to test the goals and requirements in sections *2.3.1 Goals* and *2.3.2 Requirements.*

6.1 Testing WeldPoint Remote Support (WRS) and WeldPoint Web Service (WWS)

This section describes the tests perfomed on the WRS form and its communication with WWS.

6.1.1 WRS form tests

Describes the test performed on the actual form in WRS.

Test: WeldPoint (WRS) not being connected to the $W8_2$.

Test specification: Using the WRS form when WeldPoint is not connected to the $W8_2$.

Expected Result: When entering the WRS tab in WeldPoint the form should be disabled and it should not be possible to enter any information or click any buttons.

Result: Success! It is not possible to enter any information or click any buttons.

Test: Form input test.

Test specification: Trying to enter more input characters than allowed.

Expected Result: WRS should not allow more input characters than the maximum length of a field in the form.

Result: Success! WRS did not allow more characters than the maximum length of a field in the form.

Test: Sending larger files than 32 MB.

Test specification: Trying to add larger files than 32 MB in the file list view (see *Figure 8*, page *21*).

Expected Result: WRS should issue an error message saying it is not allowed to add files larger than 32 MB.

Result: Success! When trying to add files larger than 32 MB the system issues an error messages saying “WeldPoint Remote Support does not allow case-files larger than 32 MB. The file you are trying to add is X MB.”

Test: Sending files from a USB flash drive.

Test specification: Try to send files from a portable USB memory.

Expected result: The files should be added with the button *“Add more files”* in WRS. A browse-files dialog is opened and the files can be added by browsing the USB flash drive. The files are sent as usual when clicking the button *“Send to WeldPoint Support”*

Result: Success! The files are successfully added and sent.

6.1.2 WRS and WWS communication tests

The following section describes the tests performed on the communication between WRS and WWS.

Test: WRS trying to send a case to WWS behind a firewall.

Test specification: This test is performed in the following environment:

A WRS customer using a Windows XP/Vista computer that is running WeldPoint. This computer has ZoneAlarm firewall installed and is configured to only allow outgoing traffic on port 80.

WWS running on a web server (IIS) that is running on a Windows 2003 server. This server allows incoming traffic on port 80.

The customer and the web server are not located in the same network. They communicate through the Internet. The WRS customer sends a case to WWS.

Expected result: The case should arrive at WWS and stored in the database.

Result: Success! The whole case is sent to server and uploaded to the database. This test was run both with WRS running Windows XP and WRS running Windows Vista.

Test: Two WRS applications trying to send cases to WWS concurrently.

Test specification: This test is performed in the following environment.

A WRS customer called customer 1, using a Windows XP/Vista computer that is running WeldPoint. This computer has ZoneAlarm firewall installed and is configured to only allow outgoing traffic on port 80.

A WRS customer called customer 2, using a Windows XP/Vista computer that is running WeldPoint. This computer has ZoneAlarm firewall installed and is configured to only allow outgoing traffic on port 80.

WWS running on a web server (IIS) that is running on a Windows 2003 server. This server allows incoming traffic on port 80.

The customers and WWS are all located in different networks. They communicate through the Internet. Customer 1 and customer 2 each send a case concurrently to WWS.

Expected result: The two cases should be sent to WWS and uploaded to the database.

Result: Success! The two cases are received by WWS and stored in the database.

Test: WRS communicating without a certificate.

Test specification: WRS trying to connect to WWS without supplying WWS's X.509 public certificate.

Expected result: WRS should not be able to connect to WWS.

Result: Success! WRS gets the following error message from WWS: "EncryptedKeyToken is expected but not present in the security of the incoming message."

Test: WRS trying to communicate with WWS using another certificate than the server's.

Test specification: WRS trying to communicate using another X.509 public certificate other than the server's.

Expected result: WRS should not be able to communicate with WWS.

Result: Success! WWS does not allow WRS to communicate with WWS.

Test: WRS customer cancelling a transfer.

Test specification: WRS customer clicks the *Cancel* button in the middle of a case transfer.

Expected result: WWS should remove the files already sent (if any) associated to the case and cancel the transfer. The case should not be uploaded to the database.

Result: Success! WRS cancels the transfer. WWS removes the file already sent (if any) associated to the case. The case is not uploaded to the database.

6.2 Testing of the database and WeldPoint Web Service (WWS)

This section describes tests between the database and WWS.

6.2.1 Test: WWS sending larger data than allowed.

WRS will never send larger input data than allowed by the form. But if a user manages to connect to WWS without WRS, then the WRS form restrictions will not work. This test will test if the database allows unlimited amount of input from WWS.

Test specification: Trying to send a case from WWS to the database with each system information string larger than the allowed input in WRS.

Expected result: The database should deny the case and should not store it.

Result: Success! The case is not stored.

6.3 Testing WeldPoint Remote Support Center (WRSC) and the database

This section will describe different test of WRSC.

6.3.1 Test: Incoming case, one WRSC application running

Test specification: A customer is sending a case from WRS and an WRSC application is running at ESAB.

Expected result: When the case has been stored in the database, WRSC should be notified by the database that a new case has been received.

Result: Success! The second the case has been uploaded from WWS to the database, WRSC is notified that a new case has been received. Then the case is downloaded from the database and displayed to the user in WRSC.

6.3.2 Test: Incoming case, two WRSC applications running

Test specification: A customer is sending a case from WRS and two WRSC applications are running at ESAB.

Expected result: When the case has been stored in the database, the two WRSC applications should be notified by the database that a new case has been received.

Result: Success! The second the case has been uploaded from WWS to the database it notifies the running WRSC applications that a new case has been received. Then the case is downloaded from the database and displayed to the respective users in each WRSC application.

6.3.3 Test: Deleting a case, two WRSC applications running

Test specification: One of WRSC's users is deleting a case.

Expected result: The case should be removed and the database should notify the WRSC applications.

Result: Success! The case is removed from the database and the database notifies the WRSC applications and the case is removed from both applications.

6.3.4 Test: Changing a case's status, two WRSC applications running

Changing a case's status means setting it to solved, unsolved or assigned to a user.

Test specification: One of WRSC's users is changing the status of case.

Expected result: The status of the case should be sent and both WRSC applications should be updated.

Result: The status is changed and the WRSC applications are updated immediately.

This chapter has shown that all requirements and goals have been met. The applications have been tested both individually and as whole system to ensure functionality.

7 Discussion and conclusions

The purpose with this project thesis was to solve the problem of transferring welding system information from a customer to ESAB via the Internet. The first step was to analyze different methods to send this information and then implement one of the suggested solutions. The goals also stated that two GUIs should be implemented to send and view the system information.

First off the customer's welding system was studied, which was followed by the analysis phase. To determine what unit in the welding system that should send the information was fairly straight forward when I had studied the welding system. WeldPoint was the obvious choice because of the already implemented communication with the $W8_2$. If this communication had not existed this project thesis project could look a lot different. The most time consuming part of the analysis phase was to determine how to send the system information. I would have liked to analyze a few more methods to send the system information, but there was not enough time. Web services were chosen mainly because of its feature to easily communicate across firewalls and other network obstacles since this was one of the requirements. Web services require some extra functionality at the receiver such as a web server, although the other solutions also required some kind of servers. IIS was chosen as web server mainly because ESAB uses it and that it is easy to deploy web services on it. I had never used IIS before and it took a while getting used to.

The graphical design of the two GUIs, WRS and WRSC, was developed with similar applications in mind. I tried to design WRS to look as much like a web form as possible. It is easier for the customer to understand an application if he or she has seen similar ones before. WRSC's graphical design was designed with the look and feel of an email application. I have not performed any usability testing on WRS or WRSC due to time restrictions. Although, WRS's and WRSC's features and designs where thoroughly discussed with my supervisor.

The implementation part was the most time consuming part of this thesis project. The sending of general information and system information was not hard to implement. However, there are several ways to send files with web services. I decided to use WSE 3.0 for both security (WS-Security) and file sending (MTOM). The reason why I chose WS-Security was because it has options to only encrypt the SOAP message. I chose MTOM because with MTOM the binary file data is sent inside the SOAP message and therefore also encrypted. To get the sending of files and security working was probably the single most time consuming part in this thesis project.

The solution that was presented in thesis project is very modular. For example, ESAB could easily change the web server used (IIS) to Apache. WRS could also be replaced with another application that could communicate with WWS, and the rest of the solution would still work!

7.1 Conclusions

Conclusions related to each of the goals (see section *2.3 Requirement specification*) of this Bachelor's thesis are drawn in this section.

A case is a collection of case-general information, case-system information and case-files (see section *5.2.1 What information should be sent?*)

Analysis

The analysis answers the following statements:

- What unit to send the information (case).,
- How to send the information (case).
- How to store the information (case).

The analysis phase came to the conclusion that WeldPoint is the best unit in the customer's system to send the case, mainly because the communication with the $W8_2$ is already implemented. The case is sent using a web service implementation. The reason why web services where chosen was because web services can easily communicate across firewalls and other network obstacles. The most structural way to store the case is in a database.

Implement functionality to send the system information (case).

To achieve this goal I implemented WRS. WRS is implemented under a tab in WeldPoint. WRS is the only GUI in this project thesis that the customer will use. A simple form is used to extract the case-general information from the customer. The case-system information is prefilled. When the form has been filled the customer can choose whether to add case-files. The case is then sent to WWS.

Implement functionality to receive and view the system information (case).

When WRS sends the case, it is received by WWS. WWS is implemented as a web service application running on a web server located at ESAB. WWS stores the case in a local database. This case can then be viewed by an ESAB employee using WRSC. WRSC is implemented as stand alone application. WRSC will only be used within ESAB's intranet.

Determine which log files and other information to be sent.

After discussing and consulting ESAB's developers a list of case-general information and case-system information was composed (see section *5.2.1 What information should be sent?*).

Implement functionality to send files from a portable USB flash drive.

The files on the USB flash drive can be added to a case by the customer using WRS.

The information should not be sent in clear text.

The case being sent is encrypted using WS-Security and WSE 3.0. WRS authenticates itself to WWS by providing WWS's public certificate.

This chapter combined with chapter *6 Testing* shows that all the goals and requirements for this thesis have been met.

With some additional testing I believe that this solution could be deployed immediately and used with ESAB's welding systems. The WRS form is easy to understand, the information and file sending is robust and secure. WRSC is an easy and understandable tool for ESAB's employees to handle cases with.

7.2 Future work

The implementation has been tested and should not contain any bugs. However, the solution presented in this project thesis (see *Figure 7* on page 17) should be thoroughly tested by other developers than me before deploying it. Usability testing of both WRS and WRSC would probably suggest some improvements of the applications.

The authentication mechanism between WRS and WWS could be improved. The current solution only requires WRS to supply WWS's public certificate. An extra requirement such as username and password could be added to improve the authentication mechanism even further.

WRSC's functionality could be expanded. For example, implementing search functions and more advanced user management would improve WRSC's functionality.

The legal aspects of WRS should probably be revised. For example is it legal for ESAB to store personal information about a customer without the customer's consent. The customer should be able to approve that his or hers personal details are stored by ESAB.

7.3 Experiences

I really enjoyed working on this thesis. There were many problems to solve in several different areas. For example, this thesis project involved solving networking problems with web services, configuring web servers, designing Windows applications or working with databases.

I was familiar with C# before starting to work with this thesis. The previous experience I had with C# was a summer job at ESAB working as a programmer where I partly used C#. After having used C# for a couple of months I must say it is a wonderful programming language, extremely well documented, easy to learn but still very advanced and well supported via MSDN forums. The development environment I used, Microsoft Visual Studio 2005, is the best environment I have ever used; it is fast and very functional.

The one thing I am most happy to have learned during this thesis project is web services. I think web services are really interesting and I could absolutely work with web-services related projects in the future.

Planning the thesis is always important. During the first week working with this thesis I wrote a planning report which was a requirement from the examiner. This was a good tool during the whole process of the thesis. It forced me to gather information, plan and examine the problem instead of just jumping into it.

During this thesis I have also learned to work more independently. I have had to solve problems on my own and study literature and specifications.

Bibliography

[1] Behrouz A. Forouzan (2005). TCP/IP Protocol Suite. Third Edition. McGraw-Hill. ISBN 0-07-296772-2.

[2] Wikipedia (2009). Subnet. Website. http://en.wikipedia.org/wiki/Subnetwork, Accessed 2009-04-02.

[3] NXP (2009). LPC2468. Website. http://www.standardics.nxp.com/products/lpc2000/lpc24xx/. Accessed 2009-04-03.

[4] John Sharp (2006), Microsoft Visual C# 2005 steg för steg. Första tryckning. Pagina. ISBN 91-636-0886-3

[5] Microsoft Developer Network (2009). Website. http://msdn.microsoft.com/sv-se/library/z1zx9t92(en-us,VS.80).aspx. Accessed 2009-04-03.

[6] Richard Blum (2006). C# Network Programing. Sybex. ISBN 0-7821-4176-5

[7] Microsoft Developer Network (2009).NET Framework Developer Center). Website. http://msdn.microsoft.com/en-us/library/ms229335(VS.80).aspx. Accessed 2009-04-03.

[8] Wikipedia. Java Programming Language. Website. WikiMedia (2009). http://en.wikipedia.org/wiki/Java_(programming_language). Accessed 2009-04-03.

[9] Behrouz A. Forouzan (2005). TCP/IP Protocol Suite. Third Edition. McGraw-Hill. ISBN 0-07-296772-2.

[10] George Coulouris, Jean Dollimore, Tim Kindberg (2005). Distributed Systems – Concepts and design. Fourth edition. Addison-Wesley. ISBN 978-0-321-26354-4.

[11] Web Services Glossary (2004). Web services. Website. http://www.w3.org/TR/ws-gloss/. Accessed 2009-04-20.

[12] ESAB (2009). Website. http://www.esab.co.uk/gb/en/about/Brief-facts.cfm. Accessed 2009-04-21.

[13] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson (2008). Web Services Platform Architecture. Fifth printing. Prentice Hall. ISBN-10 0-13-148874-0. ISBN-13: 978-0-13-148874-8.

[14] Munindar P. Singh (2005).The Practical Handbook of Internet Computing. Chapman & Hall/Crc Computer and Information Science Series. ISBN 1-58488-381-2.

[15] Wikipedia. XML. Website. WikiMedia (2008), http://en.wikipedia.org/wiki/XML. Accessed 2009-05-02.

[16] Wikipedia. Serialization. Website. WikiMedia (2009), http://en.wikipedia.org/wiki/Serialization. Accessed 2009-05-13.

[17] Microsoft TechNet (2008). Website. http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/848968f3-baa0-46f9-b1e6-ef81dd09b015.mspx?mfr=true. Accesssed 2009-05-15.

[18] Netcraft April 2009 Web Server Survey (2009). Website. http://news.netcraft.com/archives/2009/04/06/april_2009_web_server_survey.html. Accessed 2009-05-19.

[19] Virtualbox (2009). Website. http://www.virtualbox.org/wiki/VirtualBox. Accessed 2009-05-19.

[20] Wireshark (2009). Website. http://www.wireshark.org/. Accessed 2009-05-19.

[21] Microsoft Developer Network (2005). Website. http://msdn.microsoft.com/en-us/library/ms977317.aspx. Accessed 2009-05-19.

[22] Microsoft Developer Network (2005), Stored Procedure Basics. Website. http://msdn.microsoft.com/en-us/library/ms345147.aspx. Accessed 2009-05-22.

[23] Microsoft Developer Network, DataTable class. Website. http://msdn.microsoft.com/en-us/library/system.data.datatable(VS.80).aspx. Accessed 2009-05-24.

[24] Microsoft Developer Network, SqlDependency Class. http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldependency(VS.80).aspx. Accessed 2009-05-24.

[25] ZoneAlarm (2009). Website. http://www.zonealarm.com/security/en-us/zonealarm-pro-firewall-anti-spyware.htm. Accessed 2009-05-24.

[26] Richard Zurawski (2005). The Industrial Information Technology Handbook. CRC Press. ISBN 0-8493-1985-4.

[27] World Wide Web Consortium (W3C) (2004). Website. http://www.w3.org/TR/xml-infoset/#intro. Accessed 2009-06-08.

[28] Microsoft Developer Network (2007). Web Services Security Specifications Index Page. Website. http://msdn.microsoft.com/en-us/library/ms951273.aspx. Accessed 2009-06-08.

[29] James F. Kurose, Keith W. Ross (2008). Computer Networking, A top-down approach. Fourth addition. Pearson Education. ISBN-13: 978-0-321-51325-0. ISBN-10: 0-321-51325-8.

[30] NET Framework Home Page .NET Framework. Website. http://msdn.microsoft.com/en-us/library/w0x726c2(VS.80).aspx. Accessed 2009-06-08.

[31] Microsoft Developer Network (2006). SQL Server Management Studio Express. Website. http://msdn.microsoft.com/en-us/library/ms365247(SQL.90).aspx. Accessed 2009-06-08.

Appendix A

WSDL file for WWS. WWS's WSDL file describes what services WWS provides, where WWS is located and how to communicate with it.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.esab.com/WWS"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www.esab.com/WWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified"
targetNamespace="http://www.esab.com/WWS">
      - <s:element name="SetCaseInfo">
        - <s:complexType>
          - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="info"
type="tns:CaseInfo" />
            <s:element minOccurs="0" maxOccurs="1" name="fileList"
type="tns:ArrayOfString" />
            <s:element minOccurs="0" maxOccurs="1" name="clientID"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      - <s:complexType name="CaseInfo">
        - <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="CaseID"
type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="CompanyName"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="EmployeeName"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="EmployeeEmail"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="EmployeePhone"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Subject"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Description"
type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CaseType"
type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="IP"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Host"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Name"
type="s:string" />
```

```
          <s:element minOccurs="0" maxOccurs="1" name="Version"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="UnitID"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="MachineID"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="HwID"
type="s:string" />
        </s:sequence>
      </s:complexType>
      - <s:complexType name="ArrayOfString">
        - <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="string"
nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
      - <s:element name="SetCaseInfoResponse">
        - <s:complexType>
          - <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
name="SetCaseInfoResult" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      - <s:element name="GetUniqueClientID">
        <s:complexType />
      </s:element>
      - <s:element name="GetUniqueClientIDResponse">
        - <s:complexType>
          - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetUniqueClientIDResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      - <s:element name="AppendChunk">
        - <s:complexType>
          - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="fileID"
type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="buffer"
type="s:base64Binary" />
            <s:element minOccurs="1" maxOccurs="1" name="Offset"
type="s:long" />
          </s:sequence>
        </s:complexType>
      </s:element>
      - <s:element name="AppendChunkResponse">
        <s:complexType />
      </s:element>
    </s:schema>
  </wsdl:types>
  - <wsdl:message name="SetCaseInfoSoapIn">
    <wsdl:part name="parameters" element="tns:SetCaseInfo" />
  </wsdl:message>
  - <wsdl:message name="SetCaseInfoSoapOut">
    <wsdl:part name="parameters" element="tns:SetCaseInfoResponse" />
  </wsdl:message>
  - <wsdl:message name="GetUniqueClientIDSoapIn">
    <wsdl:part name="parameters" element="tns:GetUniqueClientID" />
  </wsdl:message>
```

```
  - <wsdl:message name="GetUniqueClientIDSoapOut">
    <wsdl:part name="parameters"
element="tns:GetUniqueClientIDResponse" />
  </wsdl:message>
  - <wsdl:message name="AppendChunkSoapIn">
    <wsdl:part name="parameters" element="tns:AppendChunk" />
  </wsdl:message>
  - <wsdl:message name="AppendChunkSoapOut">
    <wsdl:part name="parameters" element="tns:AppendChunkResponse" />
  </wsdl:message>
  - <wsdl:portType name="MTOMSoap">
    - <wsdl:operation name="SetCaseInfo">
      <wsdl:input message="tns:SetCaseInfoSoapIn" />
      <wsdl:output message="tns:SetCaseInfoSoapOut" />
    </wsdl:operation>
    - <wsdl:operation name="GetUniqueClientID">
      <wsdl:input message="tns:GetUniqueClientIDSoapIn" />
      <wsdl:output message="tns:GetUniqueClientIDSoapOut" />
    </wsdl:operation>
    - <wsdl:operation name="AppendChunk">
      <wsdl:input message="tns:AppendChunkSoapIn" />
      <wsdl:output message="tns:AppendChunkSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  - <wsdl:binding name="MTOMSoap" type="tns:MTOMSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="SetCaseInfo">
      <soap:operation soapAction="http://www.esab.com/WWS/SetCaseInfo"
style="document" />
      - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    - <wsdl:operation name="GetUniqueClientID">
      <soap:operation
soapAction="http://www.esab.com/WWS/GetUniqueClientID"
style="document" />
      - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    - <wsdl:operation name="AppendChunk">
      <soap:operation soapAction="http://www.esab.com/WWS/AppendChunk"
style="document" />
      - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  - <wsdl:binding name="MTOMSoap12" type="tns:MTOMSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
/>
```

```
    - <wsdl:operation name="SetCaseInfo">
      <soap12:operation
soapAction="http://www.esab.com/WWS/SetCaseInfo" style="document" />
      - <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    - <wsdl:operation name="GetUniqueClientID">
      <soap12:operation
soapAction="http://www.esab.com/WWS/GetUniqueClientID"
style="document" />
      - <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    - <wsdl:operation name="AppendChunk">
      <soap12:operation
soapAction="http://www.esab.com/WWS/AppendChunk" style="document" />
      - <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      - <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  - <wsdl:service name="MTOM">
    - <wsdl:port name="MTOMSoap" binding="tns:MTOMSoap">
      <soap:address location="http://localhost/WWS/Service.asmx" />
    </wsdl:port>
    - <wsdl:port name="MTOMSoap12" binding="tns:MTOMSoap12">
      <soap12:address location="http://localhost/WWS/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix B

Describes the ER diagram of the database used in this project thesis. The database contains three tables, *Case*, *File* and *User* and are all normalized in BCNF.