Web Services
- A Comparative Study

Marcus Ekelund
Andreas Larsson
**ABSTRACT**

This Master thesis enhances an existing library search system with a new dynamic web interface and introduces the ability to perform remote searches by using Web Services. The remote search functionality also includes the ability to compress SOAP-messages, this functionality is optional. The system is implemented using the Microsoft.NET Framework with Visual Studio C#. A library needs to store information about articles and we have chosen to use a Sybase Adaptive Server Anywhere database.

The practical part of this thesis describes the architecture of our implementation and the design process.

The theoretical part is a comparative study between dynamically created web pages and static pages, SOAP-compression versus no compression, SOAP-optimization versus no optimization and the use of Web Services locally versus local classes.

**Keywords:** Web Services, SOAP, UDDI, Microsoft.NET, MikroMarc, Bibliotekscentrum, Compression, ASP.NET
Preface

This master thesis in computer science has been written at the School of Mathematics and System Engineering (MSI) at Växjö University. We would like to thank the staff at Bibliotekscentrum Sverige AB (BiC) for the opportunity to work with this thesis there, and also for your support and great company. A special thank goes to Tomas Vaedelund who acted as our tutor at BiC and came up with many great ideas. We also want to thank our tutor at MSI, Morgan Ericsson for the feedback during the creation of this thesis.

Växjö, 2004-11-03

 Marcus Ekelund
 Andreas Larsson
Table of Contents

1. INTRODUCTION .......................................................................................................................... 4
  1.1 BACKGROUND ......................................................................................................................... 4
    1.1.1 Web Services ...................................................................................................................... 4
    1.1.2 Library System (Mikromarc) ............................................................................................ 4
  1.2 MOTIVATION ............................................................................................................................... 5
  1.3 PURPOSE ................................................................................................................................... 5
  1.4 PROBLEM DEFINITION ............................................................................................................. 5
  1.5 PROBLEM STATEMENTS ............................................................................................................. 5
  1.6 OUTLINE .................................................................................................................................... 6

2. METHODS ........................................................................................................................................ 7
  2.1 THE BUILDING PROCESS .......................................................................................................... 7
    2.1.1 Introduction .......................................................................................................................... 7
    2.1.2 Specifications process ......................................................................................................... 7
    2.1.3 Implementation process .................................................................................................... 8
    2.1.4 Purchasing a realized artifact .............................................................................................. 8
  2.2 CONTROLLED EXPERIMENT .................................................................................................. 8
    2.2.1 Introduction ......................................................................................................................... 8
    2.2.2 Performing a test .................................................................................................................. 8
    2.2.3 Analysis and results .............................................................................................................. 8
  2.3 USE OF METHODS ..................................................................................................................... 8

3. TECHNOLOGIES AND TOOLS ...................................................................................................... 9
  3.1 WEB SERVICES .......................................................................................................................... 9
    3.1.1 Definitions ........................................................................................................................... 9
    3.1.2 XML (eXtensible Markup Language) ................................................................................ 9
    3.1.3 SOAP ................................................................................................................................. 10
    3.1.4 WSDL (Web Service Description Language) .................................................................... 11
    3.1.5 UDDI (Universal Description, Discovery and Integration) ............................................... 11
  3.2 DEVELOPMENT TOOLS .......................................................................................................... 11
    3.2.1 Microsoft .NET ................................................................................................................ 11
    3.2.2 Code-Behind in Visual Studio .NET .................................................................................. 12
    3.2.3 Sybase Adaptive Server Anywhere (ASA) ........................................................................ 12
  3.3 SUMMARY .................................................................................................................................. 12

4. DESIGN PROCESS .......................................................................................................................... 13
  4.1 USE-CASE MODELS .................................................................................................................. 13
    4.1.1 Borrower Use-Case .............................................................................................................. 13
    4.1.2 Administrator Use-Case ...................................................................................................... 14
  4.2 COMPONENT ARCHITECTURE .................................................................................................. 14
    4.2.1 The Web Interface ................................................................................................................ 14
    4.2.2 The Core Component .......................................................................................................... 15
    4.2.3 The BUDDI Component ..................................................................................................... 16
    4.2.4 Global Architecture ........................................................................................................... 17
    4.2.5 Design – A Summary ......................................................................................................... 17

5. IMPLEMENTATION .......................................................................................................................... 18
  5.1 DYNAMICALLY CREATED WEB PAGES .................................................................................. 18
    5.1.1 Background ......................................................................................................................... 18
    5.1.2 Problem solution ................................................................................................................ 18
  5.2 REMOTE SEARCH ...................................................................................................................... 20
    5.2.1 Background ......................................................................................................................... 20
    5.2.2 Solution overview .............................................................................................................. 20
    5.2.3 The Proxy-class .................................................................................................................... 21
    5.2.4 The Bid BUDDI (BUDDI) .................................................................................................. 22
  5.3 SOAP COMPRESSION ................................................................................................................. 23
    5.3.1 Background ......................................................................................................................... 23
    5.3.2 Introduction ......................................................................................................................... 23
6. COMPARATIVE STUDIES ............................................................................................................... 27
   6.1 COMPRESSION VS. NO COMPRESSION ......................................................................................... 27
      6.1.1 The Experiment ..................................................................................................................... 27
      6.1.2 The Time Aspect ................................................................................................................... 27
      6.1.3 The Size Aspect .................................................................................................................... 28
      6.1.4 Compression Summary ......................................................................................................... 29
   6.2 OPTIMIZATION VS. NO OPTIMIZATION ..................................................................................... 30
      6.2.1 The Experiment .................................................................................................................... 30
      6.2.2 Optimization Summary ......................................................................................................... 31
   6.3 WEB SERVICE VS. CLASSES ........................................................................................................ 31
      6.3.1 The Experiment .................................................................................................................... 31
   6.4 DYNAMIC VS. STATIC .................................................................................................................. 32
      6.4.1 The Experiment .................................................................................................................... 32
      6.4.2 Summary .............................................................................................................................. 35
7. DISCUSSION ...................................................................................................................................... 36
   7.1 COMPRESSING SOAP .................................................................................................................. 36
   7.2 OPTIMIZING SOAP ...................................................................................................................... 36
   7.3 WEB SERVICE AND LOCAL CLASSES ....................................................................................... 36
   7.4 DYNAMIC PAGES ......................................................................................................................... 37
8. CONCLUSION AND FUTURE WORK ............................................................................................... 38
   8.1 CONCLUSION .............................................................................................................................. 38
   8.2 FUTURE WORK ............................................................................................................................ 38
9. REFERENCES ....................................................................................................................................... 39
List of Figures

FIGURE 1.1 – SOAP-REQUESTS AND RESPONSES ................................................................. 4
FIGURE 2.1 – THE BUILDING PROCESS .................................................................................. 7
FIGURE 2.2 – ALTERNATIVES CONCERNING THE BUILDING PROCESS ......................... 7
FIGURE 3.1 – THE STRUCTURE OF A SOAP MESSAGE ....................................................... 10
FIGURE 3.2 – THE STRUCTURE OF A WSDL DOCUMENT ................................................... 11
FIGURE 4.1 – USE CASE FOR THE BORROWER ................................................................. 13
FIGURE 4.2 – USE CASE FOR THE ADMINISTRATOR ......................................................... 14
FIGURE 4.3 – DESIGN OF THE WEB INTERFACE ............................................................... 15
FIGURE 4.4 – THE CORE COMPONENT ................................................................................ 16
FIGURE 4.5 – THE BUDDI COMPONENT ............................................................................... 17
FIGURE 4.6 – NETWORK OF THREE LIBRARIES ................................................................. 17
FIGURE 5.1 – THE DYNAMIC TABLE IN THE DATABASE ................................................... 18
FIGURE 5.2 – A SEARCH FORM IS STORED IN THE DATABASE ........................................ 19
FIGURE 5.3 – A SEARCH FORM FOR SIMPLE SEARCH ..................................................... 19
FIGURE 5.4 – THE CREATION OF A DYNAMIC SEARCH FORM ....................................... 20
FIGURE 5.5 – REQUEST AND RESPONSE, THROUGH THE PROXY CLASS ....................... 21
FIGURE 5.6 – IMPLEMENTATION OF A CLIENT SIDE PROXY CLASS ................................. 22
FIGURE 5.7 – THE REMOTE SEARCH WITH SOAPEXTENSION ADDED ........................... 24
FIGURE 5.8 – THE LIFECYCLE OF A COMPRESSED SOAP MESSAGE ............................... 25
FIGURE 5.9 – COMPARISON BETWEEN TWO SOAP-MESSAGES .................................... 26
FIGURE 5.10 – AN EXAMPLE OF A SOAP REQUEST .......................................................... 26
FIGURE 6.1 – SOAP MESSAGE ROUTE WITH SOAP EXTENSION .................................... 28
FIGURE 6.2 – DIAGRAM - COMPRESSION AND WITHOUT COMPRESSION ...................... 28
FIGURE 6.3 – DIAGRAM - SOAP SIZES .............................................................................. 29
FIGURE 6.4 – DIAGRAM - SOAP OPTIMIZATION ............................................................... 30
FIGURE 6.5 – COMPARISON - LOCAL CLASS AND WEB SERVICE – PART I ..................... 31
FIGURE 6.6 – COMPARISON - LOCAL CLASS AND WEB SERVICE – PART II .................... 32
FIGURE 6.7 – TIME GRAPH FOR STATIC PAGE REQUESTS .............................................. 33
FIGURE 6.8 – TIME GRAPH FOR DYNAMIC PAGE REQUESTS .......................................... 33
FIGURE 6.9 – USING STORED PROCEDURE ...................................................................... 34
FIGURE 6.10 – USING STORED PROCEDURE .................................................................... 35
1. Introduction

1.1 Background

1.1.1 Web Services

Web Services is a quite new concept which can be used to call methods located on a remote server (The actual term Web Services came up in late 2000 [1]). To be able to communicate using Web Services we have to use a standardized protocol called SOAP, which is built up using an XML-schema. We also need to know where the Web Service is located using the service Universal Resource Identifier (URI). Once we know where the Service we want to use is located we also need information about the method name and the XML-schemas for the request and response of the method.

The client builds a SOAP-Request using XML and one part of the SOAP includes the requested method name and parameters for the method, if needed. The server reads the SOAP-Request and calls the appropriate method; the return value of the method is parsed into a SOAP-Response (also using XML) which is returned to the client.

1.1.2 Library System (MikroMarc)

Articles in a library system are more complex than the average person thinks. Most people have the idea that author and title are sufficient information for this kind of system. But the truth is far from that. In order to have a user-friendly system where you can specify your searches to a maximum and to reduce search results we need a lot more information about each article. Examples of these attributes needed are author, title, publishing year, classification, categorization and much more. The books are stored in a Machine-Readable Catalog format (MARC) [2] which is an accepted standard worldwide.

The database used in the old system, MikroMarc [3], is a complex and well structured database. The search terms are predefined in the database using stored procedures. There exist different search terms such as author, year, classification and so on. The structure of the database is based on these search terms and each search term is represented by a unique table called index entry. Each of these tables contains the unique id for an article and some mapped search words. Therefore, if a user wants to search for an author, the correct index entry is chosen and the articles matching the search words are selected. After that we can get additional information about the selected articles from the main table in the database. The main table contains all information about all articles, more than 60 attributes and for a medium sized library more than 100 000 articles. Due to the size of the main table it would take too long to perform searches only using this table, this is why there are predefined search terms using separate index entries.
When a request for author and publishing year is triggered, the system first selects the correct index entries for year and author. When the index entries have been located the search terms are matched with those existing in the selected index entries. The result is then joined with the main table and the system can now show all information about the found articles.

1.2 Motivation
As mentioned before, Web Services technology still is quite a new concept in computer science and there are few studies available regarding web service performance and SOAP-messaging. We also want to analyze how well .NET platform handles dynamically created web interfaces.

An opportunity for us to enhance an existing, web based, library search system arose. We considered this to be an excellent system to use for us to analyze how web services function in reality with large databases, high load on the server, invoking remote web services during runtime and using dynamically created pages. Dynamically created pages are necessary because libraries are often public and different users may want it displayed in a chosen language for instance. Each library is also unique, for instance every library does not consist of books only. Therefore it could be useful for each library to have the opportunity to adapt their search pages according to their own needs.

1.3 Purpose
The purpose of this thesis is to present and compare different implementations to find out whether it is useful to utilize web service technology or not. When using this technology can we somehow optimize the SOAP-messages (for instance compressing SOAP-messages) and motivate if it is necessary.

We also want to compare static web pages with dynamically created web pages. It is extremely convenient to have dynamic pages since we want the user of the system to be able to change the way a page is shown. For instance a library administrator may want to change the search forms and a user may want the page to be shown in a different language. We want to test if this approach is affordable.

1.4 Problem Definition
A library search system already exists, using an old technology and the existing system is not that scalable or easy to maintain. Our major goal is to see if we can enhance this system with Web Service technology. How can we make this newer system more scalable, more dynamic and more user-friendly. This major goal is divided into a couple of sub problems described in the next section.

1.5 Problem Statements
From our purpose and problem definition we identified a couple of sub problems which needed to be solved in order to make a reliable and valid analysis of the system.

First we need to find a suitable way to create dynamic (i.e. Dynamic search forms) web pages; we must be able to easily change nearly everything on a page. This includes font styles and size, colors, which search fields that should be visible, what language the page should be shown in and the attributes for the text fields in the search form.
To be able to invoke methods on a remote server we need to have a reference to that server. If we do not have a reference we can not invoke the correct method on the remote server. That creates another issue which must be solved. How can we, during runtime, create a reference to a remote library?

As mentioned above, SOAP-messages are built up using XML-schemas which is easy to understand because of the structure of the document, but the disadvantage is that they tend to be very large. The actual call to invoke a method is not really an issue here since the SOAP-request will only contain which method to invoke and some parameters. But since we are building a library search system for large library databases, the result of a question can be very large. Every tuple in the result set need to have additional information (not only title of the item) such as author, publishing year, classification and other information which each library can choose. Due to the size of the SOAP-message (the response) we want to investigate if we somehow can compress the SOAP-message and what affect this will have, is it affordable?

1.6 Outline
Chapter 2 describes the different methods used in this thesis.

Chapter 3 gives an overview of the Web Service concept and will also cover development tools that have been used.

Chapter 4 briefly describes the architecture of the system so the reader can get an overview of the system. In fact the design process which are based on the Rational Unified Process (RUP).

Chapter 5 covers the implementation process and this chapter describes how the problem statements have been solved.

Chapter 6 covers four comparative studies which are derived from the problem statements.

Chapter 7 is based on chapter 6 and it discussed the results from the comparative studies.

Chapter 8 will include our conclusions for this system and also some suggestions for future work.
2. Methods
This chapter will cover the different methods applicable to this thesis. They concern two different parts, the implementation part and the test part. One constructive method and one test case method will be used.

2.1 The Building Process [24]
2.1.1 Introduction
This method is based on that an initial state is known and we want to achieve a target state. The motivation can be that there is no existing system or that an existing system needs to be enhanced.

![Figure 2.1 – The Building Process [25]](image1)

The target state can be known or unknown and there are different paths to reach this target state which depends on if it is known or unknown. If the target state is known, the goal is to implement the preferred change and if the target state is unknown the following two alternatives exist. In the first case we can start to specify the target state and after that start the implementation or we can specify and implement in parallel.

![Figure 2.2 – Different alternatives concerning the building process [26]](image2)

2.1.2 Specifications process
This process is used to realize a specification of the target state. According to Davis [27] there exist four different strategies to specify the requirements.

1. Asking
2. Derive from an existing system
3. Synthesis from characteristics of the utilizing system
4. Discovering from experimentation with an evolving system

The last strategy applies to the path where specification and implementation is performed in parallel.
2.1.3 Implementation process
When the initial state and the target state are known the next consideration is how to reach the target state. There are two alternatives to solve this consideration. The first one is to divide the problem into sub problems and if these sub problems can not be solved they are in turn also divided into sub problems. This dividing of problems will continue until we are able to solve all sub problems. The second alternative is to travel from the initial state to the target state in a sequence of transitions.

2.1.4 Purchasing a realized artefact
Buying a component is an alternative instead of creating a new one. If an existing artifact for the purpose of the system already exists, it may be unnecessary to create one since this takes valuable time and it may be more profitable to use the existing artifact.

2.2 Controlled Experiment [28]
2.2.1 Introduction
This method means that experiments are made so that as many factors as possible regarding an experiment should be under the control of the tester. There exist two kinds of variables and these are dependent or independent variables. An independent variable is for instance the programming language and a dependent variable could be the time it takes for a method to be executed.

2.2.2 Performing a test
First of all a test environment needs to be established and as many factors as possible needs to be considered. What part to test and how to gather the test results needs to be decided. After this is done the test can be performed.

2.2.3 Analysis and results
When the test is finished and all the data is gathered the result is ready to be analyzed. Aspects to consider in this part are to find a suitable way to present the results to a reader. For instance, in a comparative study, all results needs to be presented and a motivation for the result should exist which explains why one of the tested parts where better than the other part. Suitable presentations could be diagrams and tables showing the difference in performance or time.

2.3 Use Of Methods
In this thesis we will follow the previous described methods. For the implementation part of this thesis the building process will be used. The initial state will be derived from the existing system MikroMarc and implementation and specification will be a concurrent process. Some realized artifacts will be used and these are the database that belongs to the existing system and a package which contains functionality for compression and decompression.

For the comparative studies the controlled experiment method will be used. We will try to take as many factors as possible into consideration during the test part and we will present the results in form of diagrams and tables. The results will also be discussed and motivated.
3. Technologies and Tools

In this chapter we are going to explain what Web Services are and why it is an appropriate technology in these kinds of systems. We also want to give an understanding about the Microsoft .NET Framework; the capabilities and the development environment.

3.1 Web Services

Web Services are the new breed of web application. A Web Service is self-contained, self-describing, modular applications that can be published, located and invoked by other software systems across the web [20]. With this approach it is easier for software systems to communicate with each other. The software-to-software communication is exactly what we want to achieve. The communication is realized through the XML-based SOAP protocol which is sent over standard Internet Protocols, for example Hyper Text Transfer Protocol (HTTP). The Web Service architecture consist of four main parts; Extended Markup Language (XML), SOAP, Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). In the subsections in this chapter we will briefly describe these four main parts of the Web Service architecture.

3.1.1 Definitions

The concept of Web Services has already been described but we will also present a couple of definitions. There exists not only one definition of this technology, and all of them are not the same signification. The reason could be that each and every company that works with Web Services also has their own definition to describe what a Web Service really is. They may also use the Web Service technology for different purposes and that could be one reason why these definitions differ. We will present two of these definitions.

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards”.[4]

“A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. The definition of a Web service can be exported to a file, published to a lookup service, and discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols“. [5]

3.1.2 XML (eXtensible Markup Language)

Nearly the whole Web Service architecture is built up based on XML [6]. XML is a Meta language with well defined syntax and semantics. The syntax and semantics, self-describing, features of XML make it a simple, powerful, mechanism for capturing and exchanging data between applications [22]. These applications can be written in different languages and can exist under different platforms and operating systems. This since every computer can interpret plain text.
### 3.1.3 SOAP

SOAP is a lightweight protocol based on XML and is the channel where the communication between a Web Service and a consumer takes place. Or even between two or more Web Services. A SOAP message can be either a response (result back from the server) or a request (a request from a client).

A SOAP message contain four parts; envelope, header, body and fault. The envelope is required since it is the part that identifies the XML document as a SOAP message. Within the envelope tag there would be a header tag were extended info can be stored, this tag is optional. The body tag is the place where the request and response data are stored, i.e. which method to invoke at a remote server or a set of results if it is a response. Within the body tag a fault tag may exist which will be present if there had been any errors during processing the message.

These SOAP messages are based on plain text, in fact XML. Since SOAP messages are based on XML it will be easy to communicate with SOAP within a heterogeneous network, where Web Services can be found at various internet web servers. There are even possible to maintain communication between a client written in Java and a Web Service written in .NET framework since XML is text based.

```xml
<?xml version="1.0"?>
<soap:Envelope
 xml:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

*Figure 3.1 – The Structure of a SOAP message [21].*
3.1.4 WSDL (Web Service Description Language)

WSDL is an ordinary XML document where a Web Service is described. In the WSDL document there is information about the location of the Web Service and specifications of the method that are present [23]. The method specification includes parameters and the return type of the method. When a Web Service is published it is actually the WSDL-documents that are published, not the functionality of the Web Service.

![Figure 3.2 – The Structure of a WSDL document [23].](image)

3.1.5 UDDI (Universal Description, Discovery and Integration)

UDDI is a registry where Web Services are stored. In fact you may compare the UDDI with the traditional yellow pages which were a way to be able to store phone numbers and the information about the owner of the number. The UDDI works in the same manner. At the UDDI there is support to register, update and search for Web Services.

3.2 Development Tools

Within this section we will describe the development tools and environment we have been using during this thesis. Web Services can be developed using a variety of tools for example Java and a Tomcat server or using Microsoft Visual Studio on a Windows server.

3.2.1 Microsoft .NET

We have chosen to use development tools from Microsoft due to a demand that our system must run on a Microsoft Windows Server with Internet Information System installed.
What is Microsoft .NET?

“Microsoft .NET is software that connects information, people, systems and devices. It spans clients, servers, and developer tools and consists of:” [7]

- The .NET Framework 1.1 used for building and running all kinds of software.
- Developer tools such as Microsoft Visual Studio 2003.
- A set of servers, for example Microsoft Windows Server 2003.
- Client software, for example Windows XP.

In the .NET Framework there is support for over 20 different programming languages and it is very easy to develop powerful applications. Since the support for different programming languages is good we have to choose one. We chose to implement our web application using C# and Active Server Pages (ASP) .NET since C# is quite similar to Java and C which we have been using a lot earlier during our years at the university. The combination with C# and ASP.NET is also a powerful combination if you want to create advanced web based functions. When creating methods using C#-code you can choose to have these functions embedded in the ASP-document or you can implement them using Visual Studios code behind functionality. This code behind functionality is powerful and the reason why we chose ASP.NET instead of regular ASP. The code behind will be more discussed in the next section.

3.2.2 Code-Behind in Visual Studio .NET

When creating scripts in regular ASP you have to embed these scripts in the HTML code but with the new ASP.NET you can directly call functions in another document, which gives the programmer better readability of the implementation. With ASP.NET, each ASP page can have a class file linked and in this class file we can implement functions, instead of implementing these functions in the ASP code. This linked class file is called Code-Behind. With this code behind ability we can use everything available in the .NET Framework.

3.2.3 Sybase Adaptive Server Anywhere (ASA)

The existing system uses a complex and well structured database with Sybase ASA-7 and due to the complexity and time it would take to create a new database of this level we will continue using the old database for our system. The only time we have to spend now is trying to understand the existing component to be able to implement our system.

3.3 Summary

The use of Web Services is suitable for many reasons in these kinds of systems. First of all it is possible that libraries use different hardware and operating systems. Web Services can be used to communicate within heterogeneous networks since the communication is handled by using SOAP messages. These SOAP messages are, as mentioned earlier, based on XML, i.e. plain text. SOAP can be sent using different Internet protocols. In our approach it quite suitable to use HTTP and port 80 since the participating servers will be using firewalls, and if port 80 is used the firewalls will not need any further configurations.
4. Design Process

This chapter describes the analysis and design of the MikroMarc Web Service (MMWS) system. We will present use-cases for different actors and how the different components interact with each other in the system. The design process was made using some of the guidelines from the Rational Unified Process (RUP). Since the design process is not a major issue in this thesis we will not get into too much detail.

4.1 Use-Case Models

Our system will be utilized by two different actors with different access rights, namely the end-user who is a borrower and the administrator of the library system. These two actors interact with the system in different ways.

4.1.1 Borrower Use-Case

The borrower can interact with the web interface in four different ways:

- **Perform a simple search** – Search for articles in the local library database using a simple search form with no extended functionality.
- **Perform an index search** – Search for different subjects in the local library database.
- **Perform an extended search** – Search for articles in the local library database using an extended search form with additional functionality for instance the user can choose what type of information to search for and combine search words with an and-operator or an or-operator.
- **Perform remote searches** – Search for articles in remote library databases (Web Service connection) using a simple search form.

![Use Case for the borrower](image-url)
4.1.2 Administrator Use-Case
To be able to solve one of our problem statements, the ability to perform remote searches, we need some kind of global register where we can find information about every library using our system. Therefore we need an administrator use-case for this register. The administrator can perform two tasks:

- **Register library** – The administrator at each library need to register the information about the MMWS.
- **Update library** – If contact information changes the administrator has the ability to login to this register and change the information of the MMWS.

![Figure 4.2 – Use Case for the Administrator.](image)

**Administrator use-case description**
*Precondition:* The administrator must have received login information and then he has to log in to realize the activation and this use-case.
*Post condition:* A new library has been added to the register or an existing library has been updated.

4.2 Component Architecture
From our problem statement and our use-cases we have identified three key components which need to be implemented. We need a web interface through which the user can interact with the functionality of the system; we also need a core part wherein all functionality should be. In order to perform remote searches we also need to implement a global register which will function as a simplified UDDI (Universal Description, Discovery and Integration) [8]. This register is from now on called BUDDI (BiC UDDI).

4.2.1 The Web Interface
A user can perform several different types of searches and we also need additional functionality when a search has been made. Due to this we decided to divide these different search forms into different pages, in this way it is easier for us to implement and for a user to navigate. We also decided to present the search result in a different page since the layout differs between a remote search and a local search. A remote search should present at which libraries the articles were found.
Figure 4.3 shows how we want to divide our different pages where the default page will act as a coordinator page and will guide the user through the site. The other pages will be created within the default page because we want to avoid the use of frames in this system. This was a criteria from the beginning since the old system where using frames. We have made the web interface as a component and this component will be attached to the systems core component.

4.2.2 The Core Component
Derived from the problem statement and our use-case model we started to design the core component, from now on called the MikroMarc Web Service component (MMWS). We need a class to handle the functionality for local requests namely database searches requested from the web interface component. We also need a Web Service class that can handle incoming requests from remote libraries. To be able to perform searches on remote libraries we will need to have a client side proxy class which can create a connection dynamically to remote servers. For the purpose to see if we can compress SOAP-messages we will also need a part which can fetch and modify these messages, this class is called SoapExtension. We will discuss the SoapExtension in more detail in chapter 5.
Figure 4.4 shows how the core component will be implemented. We have four different paths through which we can communicate with this component for different purposes. The top path is where the web interface will be connected and most of the local requests will only query the database. The remote searches will fetch available libraries from a configuration file and query the selected remote libraries through the client side proxy class and the rightmost SoapExtension class. We also need have the option to force a connection to the BUDDI component to get the latest list of participating libraries. All incoming requests from remote libraries enter through the leftmost SoapExtension to our Web Service which then queries the database and returns the result.

4.2.3 The BUDDI Component
The BUDDI will be implemented as a Web Service since all libraries need to be able to connect to the BUDDI and get information about the libraries registered in the BUDDI. The BUDDI component will be a simple implementation and will consist of methods for registering and updating of libraries, this functionality is used by the administrators of the participating libraries. When a library has been added or modified the BUDDI component will have the ability to send latest the information to all participating libraries by connecting to their Web Services. Methods for fetching information about libraries will be implemented.
4.2.4 Global Architecture
The use of the component architecture will make the global system very scalable. It will be very easy to connect more and more libraries and create a network where each library can perform searches at other libraries. Once a new library has been added to the BUDDI it participates in the global system.

The library component consists of one core component and one web interface and every library component has a connection to the BUDDI. Figure 4.6 gives an example of the global system with three participating libraries and the BUDDI.

4.2.5 Design – A Summary
This was just a short description of our design process to give the reader an understanding of how we want the system to look like. In the next chapter we will discuss the major components and our implementation with focus on our problem statements which were creation of dynamic pages, referencing remote components during run-time and the compression of SOAP.
5. Implementation

In this chapter the focus will be on the components we have described in the design chapter and we will explain how we have solved the problem statements.

5.1 Dynamically created web pages

5.1.1 Background

Since every library is may have different opinions about how their search forms and pages should look like, we need to somehow generate these pages dynamically. A problem is that each library is unique; not every library consists of books or articles only, for instance a musical library. A musical library may not want their users to search for the term “author”, instead the term “composer” or “artist” may be a more suitable term to use for their users to understand the web interface. Some libraries are divided into different catalogs with music and books in different catalogs, those libraries may want the search interface to change depending on what catalog to search in.

5.1.2 Problem solution

The layout issue is easy task, since a solution using cascading style sheets already exists and is very convenient. Therefore we chose to use css-files. We also want every library to be able to change the standard language their page should be shown in and every user of the system should have the option to see the page in their own language. One solution is to create a new static page for every thinkable language, but that is not a satisfactory solution because updating of the system would be very time-consuming. Therefore, in our implementation, we load everything visible to the user from language files. When a page is requested the system determines which language to use, if a language has not been specified the standard language of the system is used. Since we load every text visible for the user from language files, we have no plain text in the implementation. Instead of plain text we have a method which loads the correct parameter from the chosen language file. This makes the system easy to maintain and translate into different languages.

One of the most important interface issues in the system are the dynamically created search forms. This is solved by fetching all information about a search form from the database, where the information is stored in a single table where each tuple describes one cell in our search table. Each search form consists of a search table with three columns and an arbitrary number of rows. Therefore we need three tuples to describe one row in the search table.

<table>
<thead>
<tr>
<th>Seq</th>
<th>Page</th>
<th>Command</th>
<th>Length</th>
<th>Type</th>
<th>SQL</th>
<th>Var</th>
<th>Visible</th>
<th>Catalog</th>
<th>listSelected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>String</td>
<td>String</td>
<td>Int</td>
<td>String</td>
<td>String</td>
<td>String</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
</tbody>
</table>

*Figure 5.1 – The dynamic table in the database*
Sequence; can not be NULL and determines in what order the elements should be placed in our search form.

Defines which page the element belongs to, can not be NULL

The search command used in the database.

The width of a text field, measured in pixels.

Type of the element, can be a text field, a text box, a dropdown list, a AND / OR dropdown, a dropdown changer or NULL.

If the Type is a dropdown, this SQL string selects all elements to be added to the dropdown list.

The variable name to fetch from the language file.

Set to 1 if it should exist on the page.

When a dropdown changer is used we can assign the elements to different catalogs, if this is zero the element will be visible on all search fields for the page, otherwise it will only be visible when the correct catalog number is selected.

Defines the preselected value in a dropdown list.

When a search page is requested the system first loads the appropriate language file and then starts to generate the search form by selecting from the dynamic table in the database and order by the sequence attribute. The screenshot of the search form in figure 5.3 is created based on the table in figure 5.2.

<table>
<thead>
<tr>
<th>Seq</th>
<th>Page</th>
<th>Command</th>
<th>Length</th>
<th>Type</th>
<th>SQL</th>
<th>Var</th>
<th>Visible</th>
<th>Catalog</th>
<th>listSelected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Simple</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>Text</td>
<td>(NULL)</td>
<td>Freetxt</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
</tr>
<tr>
<td>2</td>
<td>Simple</td>
<td>FT</td>
<td>150</td>
<td>Field</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
</tr>
<tr>
<td>3</td>
<td>Simple</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Simple</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>Text</td>
<td>(NULL)</td>
<td>Author</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
</tr>
<tr>
<td>6</td>
<td>Simple</td>
<td>FÖ</td>
<td>150</td>
<td>Field</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
</tr>
<tr>
<td>7</td>
<td>Simple</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>1</td>
<td>0</td>
<td>(NULL)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2 – Example of how a search form is stored in the database

*Enkel Sökning*

Freetxt

Författare

Sök

Figure 5.3 – A search form for simple search derived from figure 4.2

The text in figure 5.3 is fetched from the language file based on the variable name in the dynamic table. The search fields are fetched from the database and they get their ID from the attribute “command” in the database. Every single web control created is associated with the command and we need this command to know what type of search the user want to perform.
When a user requests a search page the language is determined and a method in code-behind is invoked. First the appropriate language file is loaded and another method is invoked in the local search library class. This method queries the database and returns the layout of the requested form. The code-behind translates the result and generates the search form. If a dropdown list exists in the generated search form another query has to be made to the database in order to know what elements this dropdown list should contain. When the page is generated it is returned to the user.

5.2 Remote Search

5.2.1 Background
If a user performs a search at one library and does not find what he was looking for, it would be user friendly if he can select more libraries to search in, without having to find the libraries himself. In this case it is useful to use the power of Web Services since the system can access remote libraries instead of making the user find the remote libraries on his own. With this approach the user only have to make one search at one library to get the results from an arbitrary number of libraries, instead of finding the libraries first and perform a search at each one of them.

5.2.2 Solution overview
To realize this kind of system we will need one component, our BUDDI, which is a lightweight implementation of Microsoft’s UDDI [9] where you can register and search for Web Services. In our version there is only support for registration and update of Services. The BUDDI can also connect to other libraries when the register has been updated in order to send out the updated information. This functionality will be handled automatically by the BUDDI. This kind of register is required to keep track of where the different Web Services are located. Another component required is a client-side proxy class which can create references to remote Web Services during runtime. The proxy class uses a configuration file to retrieve information about the remote service, or a connection to the BUDDI can be forced in order to update the configuration file with the latest information. These two components solve the problem with creating references during runtime and will be discussed in more detail in the next two sections.
5.2.3 The Proxy-class

This is the class that handles the actual connection to a remote MMWS. When adding a web reference in Microsoft Visual Studio .NET, a client side proxy class is added to your project. This class handles the connection to the remote Web Service. In fact the connection is established via Remote Procedure Call (RPC).

We modified an auto generated client side proxy class so that we were able to pass a URI (i.e. URI to the remote MMWS) as a parameter to the class constructor. To be able to invoke the search method at the remote MMWS, this method needs to be overloaded in the proxy class. Ordinary client side proxy classes are statically bound to a specific service but with this implementation we are able to connect to any MMWS where the URI and method name is known. Without this client side proxy class we would not be able to invoke methods on remote libraries where the URI is not known at the implementation stage. Since libraries can be added to the network in the future, it would be impossible to add static web references at the implementation stage. Figure 5.6 below shows how the client side proxy class is implemented.
5.2.4 The BiC UDDI (BUDDI)

This register contains the information that is required to dynamically bind MMWS’s to perform remote searches during runtime. The most important information is the endpoint (URI) to the MMWS because if the system does not know the URI, the remote search can not be accomplished. When a library makes the decision to allow other libraries to search at their server they need to register their MMWS at the BUDDI Client. At registration the library has to enter information like endpoint, description, whether the MMWS is public or not, version of the MMWS and region.

Some description about the attributes:

**Endpoint**
The most significant attribute, in fact the location of the MMWS, i.e. [http://myserver.com/mmwebcat/MMWS.asmx](http://myserver.com/mmwebcat/MMWS.asmx)

**Description**
This is the attribute shown to users when they want to perform searches, i.e. Växjö library

**Public**
1 indicates public and 0 indicates private.

**Version**
In the future the system may change and we need to know the version of the MMWS. Important for back compatibility.

**Country**
The country where the MMWS is located.

**Region**
Where the library is located in the above specified country.

**Username**
A username to be able to login at the BUDDI client.

**Password**
Required to be able to login at the BUDDI client.

When the BUDDI has been updated a method at the BUDDI is invoked which tries to contact all participating libraries and sends the updated information to those, in order to keep them updated. With this functionality the libraries will not have to connect to the BUDDI each time a user wants to perform a remote search, this will lead to a faster system. Before the BUDDI sends the updated information to a library it first checks to see if that library is alive and can receive the information. If there are some libraries not responding within a predefined time span, an exception will be raised so that the administrator can contact the failed libraries.
5.3 Soap Compression

5.3.1 Background
SOAP messages use Extensible Markup Language (XML) and this structure has a tendency to create quite large documents. The basis for this thesis was library search in a large database that contains many articles (100 000 – infinity). A search in a large database may create a very large SOAP message if the search criteria are too vague, and a large SOAP message will cost bandwidth.

In this system we have chosen to use Datasets [10] which is an object also built up of XML. This object will have the same structure as the result from a specific database query. The result from a query will contain the same tags over and over again but with different information for each row, due to the structure of XML which is built up of tags. For instance, if the search returns 100 rows and five attributes for each row, this will lead to a SOAP-message with one tag for each row and one tag for each attribute. There will be 500 tags for the attributes and 100 tags for the rows. As mentioned earlier the SOAP message will grow very big and in this case there are only 100 hits, more hits will lead to an even larger SOAP message.

This is the way XML-documents are built up and we have to accept it or choose another technique. We are going to stay with the XML technique because we use the Web Service technology for this thesis.

5.3.2 Introduction
As mentioned above the SOAP message size has a tendency to be quite large when the MMWS returns large datasets. This will cost the affected parties a lot of bandwidth and the SOAP response will take time to send back to the requested server. With compression we will lessen the bandwidth problem, but the response time may increase if you look at the time it takes to compress and decompress the message. On the other hand the data will travel faster over the internet wire and we decrease the use of bandwidth. We will analyze if it is worth the time it takes to compress / decompress the SOAP message or if it should be sent without compression.

Our approach is to both optimize and compress the SOAP-message. For optimization we will use aliases for the attributes that belongs in the database. For instance the attribute ‘pm_value21’ will have the alias ‘pm21’ and as you can see there will be four characters instead of ten and that is an easy and fast optimization. We will also use a zip algorithm to compress and decompress the content within the SOAP-message, i.e. all child tags that belong to the requested method tag. We can not compress the whole SOAP message; if we do that our Web Service will not recognize the message as a SOAP message and would do nothing about it. To be able to catch this SOAP message, in order to change it, we need to implement a component that can catch the message, namely a Soap Extension [11, 12].

This soap extension component inherits from a class called SoapExtension. The SoapExtension class gives us the opportunity to inspect and modify a SOAP message at specific stages in the message processing on either the server, client or at both sides. To be able to both compress and decompress the SOAP message we need to do this at both sides.
5.3.3 Soap Extension
The soap extension class we have implemented is used to catch SOAP messages. This component inherits from the SoapExtension class. In this component it is necessary to override some important methods [11]. These are:

- **ChainStream**: This method is overridden for the purpose of that we need to buffer the “old” SOAP message.
- **GetInitializer**: This method is overridden to initialize data for specific methods. For instance if a SoapExtensionAttribute [13, 14] (more information further down) is implemented we can choose to apply the SoapExtension to specific methods instead of the whole Web Service.
- **Initialize**: An overridden method that allows a SOAP extension to initialize itself using the data cached in the GetInitializer method.
- **ProcessMessage**: An overridden method that decides at which stage the message is.
First of all we need to backup the old stream (the original SOAP message). When this is complete we need to know in which stage the SOAP message is and which type of SOAP-message it is. It is either a request or response. To be able to decide in which stage the SOAP message is when we catch it we need to override the ProcessMessage method. There exist four different stages in a SOAP message lifecycle [15] which are:

**Before Deserialize** This stage occurs right before the SOAP formatted message is deserialized back into an object.

**After Deserialize** The stage just after the SOAP message has been deserialized back into an object.

**Before Serialize** The stage just before the object being serialized into a SOAP message.

**After Serialize** The stage after the object has been serialized into a SOAP message.

---

**Figure 5.8 – The lifecycle of a compressed SOAP message**

### 5.3.4 Optimization

We used an alias optimization where we altered attribute names from the database with shorter names, i.e. 'marc_id' became ‘mid’. This is an uncomplicated way to make optimization and with this approach the text within the tags stays untouched but we decrease the size of the unnecessary large XML tags. The compression of the SOAP-message is highlighted in the next section within this chapter.

As mentioned earlier in this chapter we can decrease the size of the SOAP-message with this alias optimization. Imagine how many unnecessary characters that will be sent over the internet wire if you have unnecessary long names for the attributes in the database. This is not at the same level as compression but if a search returns 20 000 hits we will save a lot of unnecessary characters. In the comparative study chapter we will present some test result and give more precise figures of how much this kind of optimization reduces the SOAP message and if it is affordable.
5.3.5 Compression

After optimization it is time to compress the data that exist within the ‘method name’ tag in the SOAP message (i.e. the tag name are the same as the invoked method name).

As mentioned earlier in this chapter we implemented a soap extension class to be able to catch SOAP messages that are sent over the wire so we can modify them. After we have caught the SOAP message we can start to compress or decompress the actual data in the SOAP. We are using an open source package, NZipLib [16, 17], which is under the GNU license. This package is implemented as an assembly for the .NET platform and is easy to use and has great performance for this purpose. The library has support for checksums, gzip, zip and more. We use the zip and checksum functionality to be able to realize compression and decompression.

Since compression will lead to one more step in the process of sending and receiving SOAP messages it will take some extra time to send and receive the SOAP message. To decrease the response time we only compress and decompress SOAP responses that are larger than a predefined size and this functionality can be turned off by an administrator. We do not compress the SOAP-requests. We decided to act like this since SOAP-Requests only contain the specific method and some parameters to pass to the method. The request will not result in a large amount of data so we do not need to compress the data; it will only be time consuming. In figure 5.10 below you can see that a SOAP-request contains a very small amount of data. In the comparative study chapter we will present some test studies that show how the compression affect response times and how much bandwidth we can save.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
>
  <soap:Body/>
  <coSearch xmlns="http://tempuri.org/"
    xsi:type="xsd:string">###</coSearch>
  <cat xsi:type="xsd:string">###</cat>
  <aSearchAt xsi:type="xsd:string">kelund</aSearchAt>
  <bib xsi:type="xsd:string">1:3</bib>
</soap:Envelope>
```

Figure 5.10 – An example of a SOAP Request

Figure 5.9 – Comparison between two SOAP-messages.
6. Comparative Studies

This chapter discusses and illustrates four different comparative studies. These four studies are Soap compression versus no compression, Soap optimization versus no optimization, the use of Web Services locally versus only local classes and finally dynamic search forms versus static search forms. The server used in all of the studies was an Intel Pentium 800 MHz dual processor, 512 Megabyte RAM using Microsoft Windows Server 2003, Microsoft .NET Framework 1.1 and Sybase SQL Anywhere version 7.

6.1 Compression vs. No Compression

In this comparison part two aspects has to be considered, the time to receive the SOAP message and the size of the SOAP message.

Time is important since users do not want to wait too long to get feedback from the system. The feedback issue is important because we want libraries to choose this system instead of other systems. In our case the feedback to the user is the dataset with the result from one or more remote servers on the web. The size of the SOAP is important of another aspect – the bandwidth. The goal is to always send the smallest amount of data as possible, over the Internet wire, and at the same time get the correct result.

6.1.1 The Experiment

The purpose is to compare the response time with the SoapExtension activated and not activated to be able to analyze the time it takes to get the result back to the client. The data gathering process included two remote MMWS with one small search that resulted in eight hits and one large search that resulted in 11 902 hits. These two search requests were repeated 30 times in order to get as reliable results as possible.

The second test case was based on the size of the SOAP message which is an interesting aspect to investigate. This test was based on six different search requests that resulted in 4 to 6389 hits. We modified the implemented SoapExtension and added a method which logged the SOAP message into a file, and these files were used to compare compressed and regular messages (uncompressed).

Both tests were performed on at Local Area Network (LAN) with a capacity of 100 Mbps. Since the speed of the network communication is high, the transmission time between MMWS and client can be neglected considering the time it takes to compress a SOAP message.

6.1.2 The Time Aspect

The response time is essential since that time is important for the user of the system. If the feedback takes too long, the user may choose another system. If SOAP compression is used, this will lead to increased time of the response since we must add an event in the SOAP message lifecycle. As mentioned earlier in this thesis we have added a SoapExtention component that examines the SOAP before it is forwarded to the destination and this extra event will probably be time consuming. Figure 6.1 [18] show how the SOAP travels when SoapExtension attached.
Our test results has shown that it takes more time to compress large SOAP messages that small ones. Figure 6.2 shows that the large search took longer time to compress. The small search resulted in a time difference less that 1 / 10 of a second when compression were used and not.

As mentioned earlier in this chapter, this comparative study is based on two searches, one small and one quite big search at two MMWS. The small search is a search at the search word “Brynäs” and returns eight results and the big search is at “Lars” which returns 11 902 results. As you can see in the diagram in figure 6.2 the small search does not differ that much between compression and no compression. But on the other hand the big search differs with about nine seconds and that is a lot of time for a user to sit and wait for feedback.

6.1.3 The Size Aspect
Since we store the results from the database in datasets, the size of the SOAP responses grow very big when the dataset is serialized into XML, this due to the XML-tags being repeated for each row in the dataset. When only a couple of rows are returned to the client the SOAP response will not be very big, but if there are for instance 2000 rows returned to the client the SOAP response will get quite large.

As mentioned before, the goal is to send as little data as possible and at the same time get the supposed result. This is exactly what we achieve when the size aspect of the
SOAP message is the priority. In figure 6.3 we present the diagram of the compressed size compared with the SOAP message uncompressed.

![SOAP Compression - Size aspect](image)

*Figure 6.3 – Diagram over SOAP sizes with compression and without compression.*

When analyzing the diagram in figure 6.3 we can see that the compression of small SOAP messages is not that noticeable compared to compression of larger SOAP messages. One point of interest is that a search that returns 6389 hits will send nearly 2 megabytes of data over the internet wire. These figures just represent one search from one user, and then imagine if there are 100 users that perform searches at the same time and each user search for the same articles. This scenario leads to that the remote server has to send 200 megabytes at one time and that will cost bandwidth both for the remote server and the server the users perform their searches at.

The same scenario with compression will only lead to that the remote server sends 30 megabytes totally, which is a great improvement.

### 6.1.4 Compression Summary

The studies in this section prove that with SOAP compression we will save a lot of bandwidth which is extraordinary. But at the same time compression will delay the feedback to the user quite much. These test results arises a couple of questions.

- Is it necessary to compress SOAP?
  - The broadband getting faster and faster, this gives us more bandwidth.
- Is there any hybrid of these time and size aspects?
  - Is there another way to solve this?
- Which is most important; reduce the use of bandwidth or a fast feedback to the user?

These questions are more discussed in the discussion chapter. In that chapter we will discuss and give our opinions about which solutions that are best and most appropriate for this kind of system.
6.2 Optimization vs. No Optimization

In this part, the main consideration is how much we can reduce a SOAP message through tag optimization. With tag optimization we mean that the attributes from the database are replaced with shorter names, for instance ‘pm_value20’ where replaced with ‘pm20’. With this kind of optimization the soap will be reduced with 12 characters (six in the start tag and six more in the end tag) for each row in the dataset within the SOAP message. Since the tag names are replaced in the SQL-query the optimization will not take longer than a regular search.

6.2.1 The Experiment

The whole idea to compare optimized SOAP messages with un-optimized are based on how much we can reduce the size of a SOAP message – in fact the SOAP response.

To be able to gather data (SOAP responses) we utilized the implemented SoapExtension so that the SOAP message could be stored in a file. We do not compress the SOAP message. To accomplish the test we compare the optimized data and the un-optimized with each other to see difference in size. The actual test case was based on six different searches that resulted in 4 to 6389 hits. After the data gathering we analyzed the figures and tried to see if the size reduction were the same percentage for small and large searches. The test results are presented in the diagram in figure 6.4.

![SOAP optimization diagram](image)

*Figure 6.4 – Diagram that shows the difference in size between optimized SOAP messages and un-optimized SOAP messages.*

With this optimization we get smaller SOAP responses but the reduction in size is not really noticeable for small SOAP messages. For example, the search that resulted in four hits had the reduction of 5%. When the SOAP message grew, the reduction in size was larger and was established at around 15% for results with more than 400 hits. This kind of optimization is an easy way to save some extra bandwidth.
6.2.2 Optimization Summary
This study proves that SOAP optimization is a very easy and fast way to perform a minor size reduction, it takes no extra time and it is easy to implement.

- Is it necessary to optimize SOAP?
  - If we compress the SOAP too.
- Time aspect?
- Readability – understandable tags or shorter names?

These questions are more discussed in the discussion chapter. In that chapter we will discuss and give our opinions about which solutions that are best and most appropriate for this kind of system.

6.3 Web Service vs. Classes
Should the system utilize Web Service technology even for local searches, or should the use only be applied for remote searches. The reason why we want to test this is that the creation of SOAP messages may cost some extra time and those SOAP messages are not necessary when we want to invoke search methods on the local machine, in fact they are only important to create to be able to perform remote searches.

6.3.1 The Experiment
To be able to perform this test we used our local class and a Web Service class with the exact same functionality. The tests were based on 16 searches that resulted in 4 up to 3732 hits. To get the most reliable test result we decided to write a test program that so to say spammed the Web Service and the local class with these 16 searches. We spammed the Web Service and the local class with one search word and this spamming looped for 100 times. The reason for us to spam them is to get the most reliable mean value time. We divided these search results in two groups, namely, the small group and the large group. The small group resulted in 4 to 231 hits and the large group resulted in 587 to 3732 hits. We collected the time and calculated a mean value of the searches and the result can be seen in figure 6.5 and 6.6.

![Small searches](image)

Figure 6.5 – Comparison between local class and Web Service – Part I.

Page 31/40
It is shown that the creation of SOAP messages were not that time consuming as we thought from the beginning, not on small searches any way. After some calculation we found that on the smaller searches the time using the local class was 10% less than using Web Service. So the generating of SOAP is very fast in the .NET Framework. We will now present how much the reduction in time where for the large searches. The diagram in figure 6.6 shows this result.

![Comparison between local class and Web Service – Part II.](image)

The diagram shows that the percentage of the reduction in time equals to the small search test case, but since larger searches takes more time the difference is more noticeable for a user. This is the reason why we chose to use local classes instead of web services only. The use of web services is only applied for communication with the remote servers in the system.

### 6.4 Dynamic vs. Static

By using dynamic web pages it is very easy to maintain the whole web interface but the downside of this approach is that every time a user requests the page, it has to be generated. In this context we will analyze how well ASP.NET and Code-Behind generates web pages and compare this approach with a system using only static web pages.

#### 6.4.1 The Experiment

To test how well our server can serve the requests on static and dynamic web pages we used a program developed by the Apache Jakarta Project [19] which simulates a given number of users performing requests to a specific page. Our first test simulates 25 users performing 12 requests each, in an interval of one to three seconds (random timer). The specified web page will be requested 300 times. We made these tests three times for better and more reliable statistics.
Figure 6.7 – Time graph for static page requests.

Figure 6.7 shows the time it took for each request to be processed and the server load was very small. The graph shows all three tests (total of 900 requests) and we can see that it takes some time for the server to “warm up” at the start of every test. The first requests take longer time and this affects the average time.

Figure 6.8 – Time graph for dynamic page requests.

Figure 6.8 shows the dynamic request graph and here we can see the same phenomena; the server need some time to “warm up” and the requests are handled faster after the first 10 percent have been handled. The server load during this test is much higher than during the static test because of all the connections we have to make for the
database and language files. The dynamic approach leads to higher response times as seen below.

<table>
<thead>
<tr>
<th></th>
<th>Average (ms)</th>
<th>Median (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Requests</td>
<td>213.4</td>
<td>20</td>
</tr>
<tr>
<td>Dynamic Requests</td>
<td>1556.6</td>
<td>691</td>
</tr>
</tbody>
</table>

After some analysis of the result we found out that generating pages is quite fast in ASP.NET and Code-Behind in Visual Studio, the major bottleneck here are the database requests. The test page executes three different SQL queries, one to fetch the layout of the page, then there are two dropdown lists which need to be filled and they need an SQL-query each. Since the number of dropdown lists a user may want is arbitrary, the time to generate a page in this way may be much higher. Therefore we started to implement a stored procedure in our database which first gets the layout, loops through the result set and replaces the SQL-queries for the dropdown lists with the result of those queries. By using this implementation we would always have only one database access no matter how many dropdown lists a user may want. The problem here is that the database in the system does not support SQL-queries in stored procedures which return more than one row. If a newer version of the database had been used, this could have been solved. But since this was not an option we decided to make some tests were we fetch the layout of the page with a stored procedure, but we did not fetch the content for the dropdown lists.

The test without stored procedures and several database accesses each time a page is requested lead to that 41% of the requests failed. That was the reason why we chose to use stored procedures instead. The following graphs are the test results when using stored procedures.

![Figure 6.9 – Using stored procedure (25 users, 900 requests)](image-url)
By using stored procedures the average time to generate a page drops from 1500 ms down to below 400 ms when 25 users are requesting the page 12 times each, which is a good enhancement. We also performed a test with 100 users requesting the dynamic page 12 times each, this resulted in no failed requests. The response times were higher due to the high load on the server, but in comparison to the former implementation without stored procedure and 41% failure, this is a great improvement.

6.4.2 Summary
A couple of questions arose from these test cases:

- Dynamic or static pages?
  - Is it worth the extra time to gain maintainability?
  - Is it worth to upgrade the database to use stored procedures?
  - Do we need a hardware upgrade?

These questions are more discussed in the discussion chapter. In that chapter we will discuss and give our opinions about which solutions that are best and most appropriate for this kind of system.
7. Discussion

This chapter will discuss the implementation considering the test results presented in chapter 6. We will also summarize the results and give an example of a preferred implementation for acceptable performance derived from the test results.

7.1 Compressing SOAP

When using a search system we have to consider what aspect that is most important for the user; the time to get result or the cost in bandwidth for the network connections between the participating servers? This question seems to have a quite simple answer; of course the time to get feedback is most important for the user since the user does not know what happens when a search is triggered. But the cost in bandwidth is an important issue for the supplier of the system. We have to balance the demands of the user and the supplier to achieve an implementation that is accepted by both parties. Due to the technology existing today, we had to add an extra step in the SOAP messaging in order to be able to compress the SOAP message. This is not an acceptable solution since this extra step takes too much valuable time for the system to be competitive in today’s market. Since the serialization into XML is very fast in the .NET Framework a solution could be that there would exist built in functionality for compression. If this functionality had existed the extra step to compress and analyze the SOAP message would be unnecessary and therefore compression may be worth the time. In a library search system it is unnecessary to compress large search results, instead it is better to tell the user to give a more specified question. This due to the fact that a result of more than 200 hits is too vast to get a general view of, it is hard to find a specific article.

In other systems it may be useful to compress or alter SOAP messages, for instance a business system with attached files being sent using web services. In this case it could be useful to compress these messages. Or if classified information will be sent the soap extension could be used in order to encrypt the SOAP message.

7.2 Optimizing SOAP

Since there is no time cost for optimizing SOAP messages and the effect is positive, this functionality is recommended to use. If we also compress the SOAP message after the optimization this may seem to be an unnecessary step. But since the compression / decompression functionality should not exist in this kind of system, the optimization will save us some bandwidth at no cost in the response time. Another issue highlighted in the summary of the comparative study, was readability. How important is it to understand the tag names in the SOAP message for a human being? It is not important at all since the system takes care of the transformation of the SOAP message. Therefore we discarded the readability issue and we are going to use optimization since it is easy and effective.

7.3 Web Service and local classes

From the comparative studies we noticed that the actual serialization and deserialization of SOAP messages were quite fast. The first thought was that this would be a lot more time consuming than only invoking local methods where no serialization takes place. But as seen in chapter 6 the .NET Framework handles the serialization / deserialization very well though locally invoked methods executes faster. Therefore we prefer to use local classes for local searches. A guideline is to use local methods if you do not need to have that method accessible by remote systems. As we mentioned in the
implementation chapter we are using a Web Service for remote requests and a local method class for the internal functionality.

7.4 Dynamic pages

From the performed tests we see that static pages are to prefer if we want small load on the server and fast response times. If we want a user-friendly system, which is easy to update or modify, dynamic pages are to prefer even if it costs in performance. When using dynamically created web pages we must try to keep the database accesses at a minimum, only one per page request, due to the time every access takes. Stored procedures are a great solution to reduce database accesses and to be able to use this we need a newer database which in our opinion is affordable for this system.

In our opinion it is worth the extra time and money to invest in a faster server which can handle the creation of dynamic pages under high load if we want to have a system as dynamic as possible. For a smaller system where the maintainability is not that important and with information that is consistent, static pages are to prefer and then you may not need a very powerful server.
8. Conclusion and Future Work

8.1 Conclusion
After the discussion derived from our test results the following implementation should be the most appropriate for this kind of system. We need a better server than the one we used for our test cases, but we want to stay with the .NET Framework. A database upgrade is required to be able to use the dynamic pages with stored procedures. With the use of stored procedures the database accesses will be kept at a minimum and that is essential because we want to use dynamically created pages. Dynamically created pages are important if we want the system to be easy to customize for every separate user without recompiling the core component and the web interface.

Our suggested implementation will be as described in the design view with a separate component for Web Services and a separate component for the local functionality. We also want to optimize SOAP messages in order to reduce bandwidth at no extra time cost, but the compression part will be discarded since it is not necessary for a library search system. A better approach is to give the user an error message that the search request was to vague.

8.2 Future Work
We have identified three major parts that can be considered for future work. First the SOAP compression part, this part is interesting and is powerful with reducing the amount of data being sent. The drawback was that it is an extra step in the SOAP processing cycle which was very time consuming. One suggestion is to integrate the compression as an alternative when serializing SOAP messages in the .NET framework. In our opinion this may lead to a reduction in time when processing the SOAP message.

The second part is to invest in a newer database that can handle more advanced stored procedures since this will lead to better performance when generating requested pages dynamically. This future work includes the implementation of such functionality.

The last part is to extend the system to a fully functional implementation of a library search system. This system can for instance include functionality for reservation of books for a borrower, a login part, an individual profile for each user where parts such as language and different themes for a page can be adjusted. Those were just some proposals; the exact requirements are not up to us.
9. References

   http://www.webservices.org/index.php/article/articleview/1202/1/24/

   http://www.loc.gov/marc/

   http://www.bicsv.se

   http://www.w3.org/TR/2003/WD-ws-arch-20030808/

   http://www.inf.fu-berlin.de/inst/ag-tech/wsqsos/

   http://www.developer.com/services/article.php/1495091

   http://msdn.microsoft.com/netframework/technologyinfo/overview/


   http://uddi.microsoft.com/


    http://primates.ximian.com/~lluis/blog/archives/cat_web_services.html


   http://aspnet.4guysfromrolla.com/articles/012104-1.aspx

   http://jakarta.apache.org

[20] Department of Computer Science at the University of Georgia, 2002 (March, 2004)

   http://www.w3schools.com/soap/default.asp

   http://www.w3schools.com/xml/default.asp

   http://www.w3schools.com/wsdll/default.asp


[25] Pertti Järvinen, (2001) On Research Methods – Figure 5.2 page 91

[26] Pertti Järvinen, (2001) On Research Methods – Figure 5.3 page 92

