An evaluation of compression and streaming techniques for efficient transfer of XML documents with Simple Object Access Protocol (SOAP)

by
Ørjan Stallemo and Kristian Vatne

Masters Thesis in Information and Communication Technology

Grimstad, May 2003

Abstract

In SOAP, the entire XML object is generated on the server before it is returned to the client. This puts unnecessary strain on server systems in terms of both memory and CPU. The objectives are to find why SOAP does not allow streaming of responses, possible solutions to the problem and outline alternative transfer methods. Furthermore, compression techniques for a streaming SOAP environment are evaluated, as well as performance of streaming versus an alternative method of data retrieval.

The feasibility study concluded that SOAP itself allows streaming of responses, but the HTTP binding does not. This binding specifies the issue of a HTTP fault code in case of a SOAP processing error, meaning the processing must be completed before a HTTP code can legally be issued.

One alternative to streaming is using a Request/N-Response message pattern, and dividing the data over several responses. As HTTP only supports a Request/Response message pattern, implementing this is not possible.

Either the HTTP binding must be rewritten to allow streaming of responses while processing a request or HTTP must be replaced with for example DIME as the transfer protocol for SOAP to overcome these problems.

Tests are set up to find the most suitable compressor technique and to verify that streaming SOAP responses utilize server resources better then alternative transfer methods. Results show that bzip2 is the most suitable compressor technique. And that streaming utilizes memory considerably more efficient, especially with multiple clients connecting.

Preface

This thesis was written for Sense Technology and is part of the Masters degree (Master i Teknologi) in Information and Communication Technology (ICT) at Agder University College, Faculty of Engineering and Science in Grimstad Norway. The work was carried out in the period between January and May 2003.

Our supervisors have been Rune A. Skarbø (Manager E-field, Sense Technology) and Magne Arild Haglund (Assistant Professor, Agder University College), they gave us continues guidance during the entire project. We would like to thank them for their positive attitude and for assisting us. We would also like to thank Stein Bergsmark (Director of Study, Master of Science Study, Agder University College) for guidance during thesis write-up.

The source code produced during the work on this thesis can be found on a CD at the back cover.

Grimstad, May 2003

Ørjan Stallemo and Kristian Vatne

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The Extensible Markup Language (XML) has gained widespread popularity in driving Enterprise Web development today. XML promises a standard data format that can be shared easily across applications, which is especially useful for different organizations that need to share data.

Sense Technology has developed a system, Sitecom, for acquiring, distributing and managing rig-site data. This involves transferring real-time and historical data between offshore drill sites, onshore control centers, head quarter offices, remote export sites, etc.

All data is transferred as Wellsite Information Transfer Standard Markup Language (WITSML) documents, and is retrieved using SOAP. WITSML defines a number of XML objects and SOAP interfaces for accessing and updating objects. WITSML is a markup language derived from XML used by the oil industry to define a new standard for drilling information transfer.

XML has several advantages, like the act of agreeing to a common format for data is making exchange easier than it ever was before. It is an open standard that represents data in a human readable form. One disadvantage is overhead: XML documents can be many times the size of the actual data it represents.

SOAP provides a simple mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. But SOAP has no method for streaming a response while processing the request.

1.2 Thesis definition

When retrieving historical data in WITSML, a client uses SOAP to remotely query a WITSML server. In real-world applications the size of the query result is often 10-50MB or more. In SOAP, the entire XML object is generated on the server before it is returned to the client. Thi s puts an enormous burden on server systems in terms of both memory and CPU, especially when considering that servers must support multiple clients simultaneously.

XML compression tools provide different ways of compressing XML documents. These differ in compression ratio and speed, often depending on document size and structure.

XML streaming techniques is used to overcome problems with transferring large XML documents.

The objective of this thesis is to evaluate the performance of the various techniques and implementations for compressing and streaming XML through theory and testing.

Furthermore, we will explore ways to combine streaming with SOAP.

If possible, a simple test prototype will be made, demonstrating compression and streaming of XML documents in a SOAP query.

1.3 Case

1.3.1 Compression case

Usage of XML is very far-reaching. The range of industries that employ XML, in one way or another, now span over most modern industries including: oil, TV and computer programming. This is done by defining their own markup language with XML as its meta language. XML's flexibility allows virtually every industry to map their information onto XML.

This flexibility often comes at a price. In order to achieve the mapping and making it humanly readable, verbose tags are used. The results can be XML documents up to three times the size of the raw data. Needless to say, the desire to reduce the size of XML documents is prominent.

A variety of approaches has been tried in compressing XML documents. From the straight forward use of well known text compressors to analyzing the rules for structuring the document before separating, restructuring and then compressing it.

We will test the compression ratio and time used for text compressors and XML compressors on XML documents varying in size.

The techniques selected for the comparing will be selected from the single platform/environment where most techniques can undergo the testing under the same criteria.

Make a proposal, based on the test results and theory, as to what kind of compression should be used by Sitecom.

1.3.2 Streaming case

Sense Technology uses SOAP in their Sitecom System, which means the entire XML object is generated from the database on the server before it is returned to the client. This puts a large strain on server systems in terms of both memory and CPU, especially when considering that servers must support multiple clients simultaneously.

Streaming is used to overcome this problem. It allows the server to transfer the reply as it is being built, instead of storing it in memory before transfer.

We will test streaming from a database by measuring the CPU and memory load. In addition we will transfer XML objects from a server to a client. Performance will be measured by CPU load, memory usage and time used on server.

1.3.3 SOAP case

SOAP allows for applications to be invoked across the Internet, independently of platform and programming language. Applications are invoked by sending messages, often paired up as a request and a response. The use can vary from sending greetings, giving orders to retrieving information.

There is no support for streaming SOAP messages. When a request is made, the server will prepare the entire message before replying, regardless of the size of message. In some scenarios reply messages generated from data retrieved from databases can reach 50 MB. Multiple queries being processed form different clients can put large pressure on the SOAP server, both in terms of CPU and memory usage.

We will explore ways of combining streaming with SOAP, outline alternative approaches to the problem and try to find the theoretical reason to why SOAP does not support streaming.

Suggestions will be discussed, and rated by how much they deviate from the standards.

1.4 Literature review

This section is intended to show where information on the Internet relevant to this thesis can be found. The first passage is about compression, the next is on SOAP and SOAP related, and last are two articles shedding light on where streaming with SOAP is today.

http://datacompression.info/ [24] has gathered an impressive collection of resources on compression, including papers, articles, source code, executables, tutorials and news. The resources are mostly external but the site is kept up to date. One example is the paper "A block-sorting lossless data compression algorithm" [27] that was used when writing about Burrows-Wheeler Transform in Chapter 3.2.4.

The Internet Engineering Task Force (IETF) [1] describes itself as "*a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.*" IETF hosts a variety of Request For Comments (RFC) and Internet Drafts. Amongst others is "The Blocks Extensible Exchange Protocol Core" [7] that was used when writing Chapter 4.5.3.

World Wide Web Consortium (W3C) [2] develops interoperable technologies including specifications, guidelines, software, and tools. And is a forum for information, commerce and communication. W3C focus on standards that are World Wide Web (WWW) specific. The most important for this thesis was the "Simple Object Access Protocol (SOAP) 1.1" [3] specification.

For articles on where streaming with SOAP is today and problems associated with it, see articles "Using Web Services Enhancements to Send SOAP Messages with Attachments" [18] by Jeannine Hall Gailey, and "Beep BEEP!" [19] by Rich Salz.

1.5 Report outline

The target group for this report is students and E-Field engineers at Sense Technology with a basic knowledge of XML and SOAP. Readers with an interest in the problems surrounding streaming of SOAP responses may benefit from reading the report.

Chapters 2, 3 and 4 give a theoretical overview of the technologies discussed in this thesis. Each technology is explained in the detailed level needed for this thesis. For further readings, a reference to the specification or articles describing them are supplied.

Chapter 5 is a feasibility study into the possibilities of solving the problem of server resources when transferring large XML documents. We start by introducing alternatives to streaming. Then look at alternative transfer technologies that has been launched to widen the usage of SOAP as it is today, or to replace the foundation SOAP rests on. Last is an explanation to why SOAP will not stream, and a protocol layer overview.

Then methods used during the course of this thesis are described in Chapter 6, which is divided into three main parts. One part describing software solutions from outside vendors, mainly Microsoft Corporation, and on part describing the software we developed to conduct testing. The last part is about the adaptations we made going from a theoretical point to a practical.

Chapters 7 and 8 are about testing and the test results. The testing focuses on three areas: compression, database and data request over network.

Finally, in chapter 9 we discuss the feasibility study and test results, and make recommendations and suggestions to the solutions.

2 Background technologies

2.1 Overview

This chapter provides information on the technologies this thesis deals with. Most of the actual study is on technologies surrounding and related to the ones in this chapter. So the majority of this chapter will not be used in discussions or conclusions, but only servers as a way of introducing the background technologies.

First is a small introduction to the well known XML, with no rules or specification, only a short summary and an example. Next is a section that sums up the history and functions of WITSML, a markup language derived from XML. And finally, a more thorough look at SOAP.

Section 2.4 on SOAP covers an introduction, the architecture, an example, bindings and a bit about future releases.

2.2 XML

XML [5] is a set of rules for defining semantic tags that break documents into parts and identify the different parts. It is a meta-language that defines the syntax in which other markup languages can be written. To make a markup language you make up tags. These tags must be organized according to certain principles, but they are quite flexible. The tags can be documented in a Document Type Definition (DTD) or Schema.

The syntax that field specific markup languages (e.g. WITSML) follow are defined by XML. XML specifies the rule for the syntax, saying how the markup is different from content and how attributes is attached to elements. It specifies only the patterns that these elements must follow. For example, XML tags begin with a < and ends with a >, but does not tell what names that shall go between.

```
<xml>
  <element attribute_name="attribute_value">content</element>
  <empty_element attribute="attribute_value"/>
</xml>
```

Figure 2.1 Simple XML document example.

2.3 WITSML

WITSML is a standard for sending well site information in XML documents between business partners. WITSML was initially developed by a group of service companies (Baker Hughes, GeoQuest, Halliburton, Landmark and Schlumberger) sponsored by the oil industry (BP, Statoil and Shell). WITSML 1.1.0 was released in December 2001 and WITSML 1.2.0 was available as release candidate in January 2003. Active members are BP, Statoil, Shell, Baker Hughes, Halliburton, Schlumberger and NPSi.

The aim of the project is according to the WITSML official website [4]: *"The "right time" seamless flow of well site data between operators and service companies to speed and enhance decision-making".*

WITSML sends information in XML documents. The content of an XML document is based on an XML schema. The WITSML standard consists of data object and component schemas. Data object schemas is the smallest set of data that can be transported in an XML document (well, log etc). They contain attributes, elements and imported component schemas. Component schemas are XML schemas, but they do not represent complete data objects. Instead they can be implemented in several data objects.

2.4 SOAP

2.4.1 Introduction

The SOAP specification [3] defines SOAP as: *"SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML"*

SOAP is at the stage where a submission to W3C has been made to propose the formation of a working group. The current version is 1.1 [3].

2.4.2 Structure

A SOAP message is in reality an XML document that follows certain rules. The three parts SOAP Envelope, SOAP Encoding and SOAP Remote Procedure Call (RPC) have their own rules.

The rules for SOAP envelope can be found in Schema for the SOAP/1.1 envelope [47]. Figure 2.2 shows the hierarchy of SOAP Envelope.

```
<envelope>
        <header> </header>
        <body> </body>
</envelope>
```

Figure 2.2 The principle hierarchic structure of SOAP Envelope.

SOAP encoding rules define how to map features found in type systems (e.g. programming languages and databases) to XML. This includes amongst others number, date, array and struct. Rules can be found at Schema for the SOAP/1.1 encoding [48]. SOAP does not require these rules to be used, but encourages it.

Using XML in RPC is one of SOAP's design goals. In order to make a method call, information about the Uniform Resource Identifier (URI) of the target object, the method name and parameters is needed.

RPC method calls and responses are carried in the SOAP Body element, and additional relevant information that is not a part of the formal method signature is carried as a child element of SOAP Header.

Although most SOAP applications depend on Hyper Text Transfer Protocol (HTTP) for transferring their RPC, SOAP is really independent of the underlying protocol.

2.4.3 An example of SOAP message exchange

This SOAP message is from the specification [3]. The message requests GetLastTradePrice, with DIS as parameter.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2.3 SOAP message example.

2.4.4 SOAP in HTTP

The most common way of transferring SOAP messages is trough HTTP. The semantics of SOAP is defined in a way the fits to HTTP.

Although SOAP messages are one-way messages, they are often combined to form request/response sequences. These sequences fit very well to HTTP's request/response.

SOAP errors while processing a request causes the server to issue a HTTP fault code "500 Internal Server Error" response with a SOAP message containing a SOAP Fault element included.

2.4.5 The upcoming SOAP 1.2

The next version of SOAP is version 1.2. The specification is spread over the following three documents:

- Part 0: Primer [33]
- Part 1: Messaging Framework [34]
- Part 2: Adjuncts [35]

For a list of the changes between SOAP 1.1 and SOAP 1.2, please see: Section 6 in Part 0: Primer [33].

In our view, the changes are important to clarifying and improving SOAP. But they are not revolutionary in the way SOAP will be develop and used on the Internet.

In his article "SOAP 1.2 spec takes next step" [20] Paul Krill quotes W3C representative Jane Daly saying: *"The functionality of Version 1.2 is essentially the same as the existing W3C standard, Version 1.1"*

In relations to the subject of this thesis, a few new interesting aspects have arisen. From Part 2 section 6.2 about the Request-Response Message Exchange Pattern (MEP), the following is quoted:

"Bindings that implement this MEP MAY provide for streaming of SOAP responses. That is, responding SOAP nodes MAY begin transmission of a SOAP response while a SOAP request is still being received and processed."

So supporting streaming is not mandatory in the next version of SOAP, but the option is there. Hopefully the major developers will choose to implement it, even if it involves a lot of redesigning.

A "SOAP 1.2 Attachment Feature" draft has been released [36]. It describes a general way of allowing attachments, like pictures, videos and so forth, to be transferred along with the SOAP message. This is similar to how SOAP Messages with Attachments (SwA), section 5.3.1, and Direct Internet Message Encapsulation (DIME) section 5.3.2.

3 Compression

3.1 Overview

The Data compression book [31] describes data compression as follows: "Data compression seeks to reduce the number of bits used to store or transmit information"

We distinguish between two different types of data compression:

- Lossless. No information is lost during compression-decompression. An example of lossless compression is data files.
- Lossy. The decompressed data may differ from the original data. Examples of lossy compression are images compression and real time voice.

The focus in this thesis is on lossless compression of XML files.

The first section is on traditional text compressors. They are introduced with an overview of relations between the various algorithms, methods and formats. Then a short summary of how the algorithm/method/format works.

Last is a description of compressors developed specially for compressing XML documents. We start by going through the most commonly used techniques, and finish with an overview of what compressors implements which techniques.

3.2 Text compression

3.2.1 Introduction

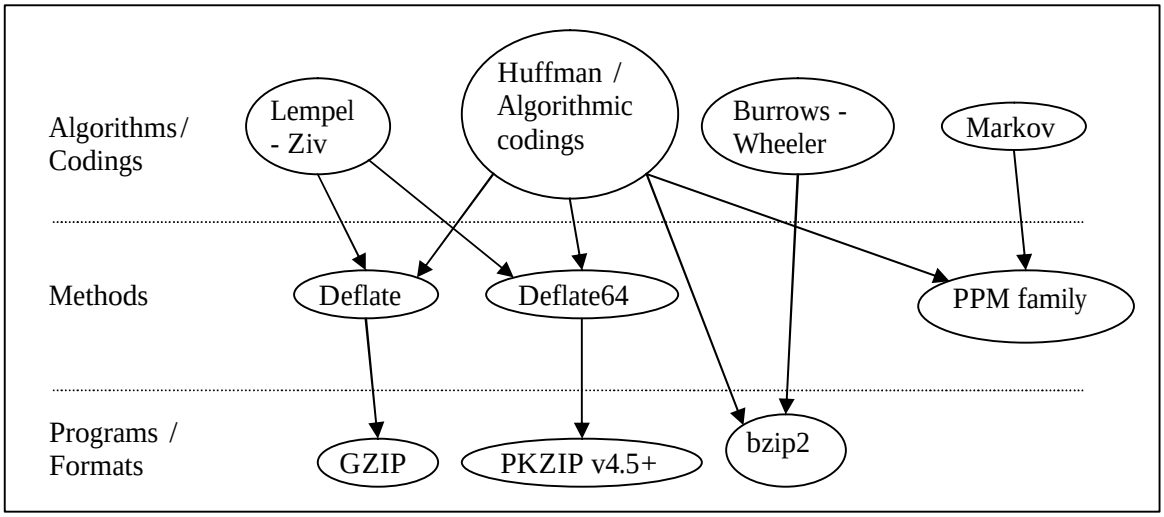


Figure 3.1 Text compression overview.

Text data compression is a form for lossless compression. Figure 3.1 shows the algorithms/methods/formats described in this section. The figure is designed to give a rough overview, not an exact picture. The borders between method, format and programs are not always straight forward.

3.2.2 Lempel-Ziv (LZ77)

LZ77 is an algorithm that uses a substitutional compression schemes proposed by Jakob Ziv and Abraham Lempel in 1977 [25]

The principle of the algorithm is to find sequences of data that are repeated. It uses previously seen text as a dictionary and replaces phrases in a text with pointers to the dictionary. The amount of compression depends on the size of the window into previously seen text and how long the text phrases are.

The LZ77 model is an attractive compression algorithm because it is simple to maintain and encode. Popular programs such as PKZip use variants of the LZ77 algorithm.

3.2.3 Huffman Coding

Huffman coding was first described in a seminal paper by D.A. Huffman in 1952 [26] and belongs to a family of codes with variable code word length.

This means that individual symbols are coded with a bit sequence with a distinct length. Huffman build on the fact that distinct symbols have distinct probability of incident. This is used to create code words that really contribute to reduce redundancy in data. Symbols with high probability are coded with shorter code words and symbols with low probability are coded with longer code words. This means that long code words do not show up as often and this contributes to the optimality of code.

Huffman code can be properly decoded because it obeys to the prefix property, which means that no code can be the prefix of another Huffman code. This compression is used in compression programs like PKZip and bzip2.

3.2.4 Burrows-Wheeler Transform (BWT)

Michael Burrows and David Wheeler released a research report in 1994 [27], discussing work they had been doing on a transformation function. The BWT transforms a block of data into a format that is extremely well suited for compression.

The basic of the algorithm is that it takes a block of data and rearranges it with a sorting algorithm, such that it is much more suited for compression. The transformation is reversible so that data can be restored with no loss of fidelity.

The idea is that if a given string of character is transformed, a few characters are likely to appear much more frequent in a distinct region. This will group up the character in the string.

The bzip2 format use this reorganized string in combination with Huffman coding for compression.

3.2.5 Markov modeling

The basics of Markov modeling is to predict the probability of a given character to appear based on what has come before it. This probability is what an arithmetic encoder, similar to Huffman, needs to perform effective compression.

The Prediction by Partial Match (PPM) [29] is a specialized form of compression based on Markov modeling.

3.2.6 Deflate

Deflate is described in RFC1951 [37]. It is a lossless compression algorithm that builds on the two compression strategies Huffman coding and LZ77 algorithm.

The compressed data consist of a series of blocks, corresponding to the input data. Each block is compressed using a combination of Huffman coding and LZ77.

The Huffman trees for each block of data are independent from the previous code blocks. The LZ77 algorithm can use reference to earlier duplicate code blocks, up to 32 KB input bytes before.

Each block of compressed data consists of two parts. A pair of Huffman code trees that describe a representation of the compressed data, and the compressed data. The Huffman code trees are compressed using Huffman coding.

3.2.7 Deflate64

The Deflate64™ , also know as Enhanced Deflate, compression algorithm is a variation of the Deflate algorithm that uses a 64 KB sliding window rather than a 32 KB window in order to compress a sequence of bits. Deflate64™ is developed by PKWARE Inc [28], and is used in PKZIP version 4.5 and newer.

3.3 XML compression

3.3.1 Introduction

In the wake of XML's success, a new category of compressors has arisen to deal with XML's verbosity. These are often referred to as "XML compressors".

XML compressors apply different techniques to in order to achieve the best compression. These techniques rearrange and code the different parts of the document. Common for all XML compressors, however, are the use of traditional text compressors in order to compress the actual data.

This section explains the most prominent techniques, and finish by giving an overview of which techniques the different XML compressors implements. Description of the implementations can be found in Appendix A.

3.3.2 Separating the document parts

XML documents consist of the element, attributes and contents as shown in section 2.2. Similarities are often found amongst the elements and amongst the attributes, and can be used to improve compression ratio. So in order to take advantage of the similarities, the XML documents are parsed, and the elements, attributes, content and document structure are separated from each other before they are compressed.

This example shows how XMill works:
<el att="123">abc</el> = [el,att];[123,abc];[S0 S1 T X T X]

"[S0 S1 T X T X]" represents the document structure.

3.3.3 Grouping related data

Once the attributes and content have been separated from the rest of the document, it can be grouped in different containers. Each container holds the data from one type of element.

```
<student>
        <name>John Doe</name>
        <number type="int">123456</number>
</student>
<student>
        <name>Jane Doe</name>
        <number type="int">123457</number>
</student>
```

Figure 3.2 XML document, example of a section.

As seen in the example in Figure 3.2, grouping all data from <name> and <number> separately will increase the chances of finding similarities when compressed.

3.3.4 DTD/Schema awareness

We divide XML compressors into two main groups, dependent upon whether or not they are DTD/Schema aware. This technique offers great advantages, and great drawbacks.

Section 2.2 outlines what a DTD and a Schema is. The compressor can utilize the knowledge from a DTD or Schema about a XML document to improve compression:

- If element A has X sub elements, then the sub elements can be coded as a list of log2 X bit. No need to code the entire sub element as the decompressor also has the list of sub elements.
- Schemas provide the data type for the document parts (e.g. a specific content are only dates). The already grouped items, see the previous section, can be compressed with a compressor specialized for dates.

The potential gain in compression ratio from using the DTD/Schema is great. XML's verbosity lies mainly in the tags rapped around the attributes. When coding the tag elements with only a few bits, this verbosity is reduced noticeably.

The major drawback with this technique is the fact that the decompressor also needs access to the DTD or Schema. Without this, it is impossible to decompress. This limits the use of the XML compressors that utilizes this technique.

It can, however, be argued that most use of XML documents that need compression often have permanent sender/receiver and follow certain DTDs/Schemas.

3.3.5 Concurrent compression

In cases where multiple XML documents are compressed simultaneously, similarities between the documents can be used to improve the compression ratio. The more similarities, the more compression ratio improves.

A precondition for decompression XML documents compressed with concurrent compression is that all documents are received before decompression. This limits the range of use.

3.3.6 Implementations overview

Description of the XML compressors can in found in Appendix – A

Table 3.2 shows what techniques some of the XML compressors have implemented.

Table 3.1 XML compressors and implemented technique.

XML compressor \ technique	Separating document parts	Group related items	DTD / Schema awareness	Concurrent compression
XMill	X	X		
XMLPPM	X	X		
Millau	X	X	X	
XComprez	X	X	X	
XML-XPress	X	X	X	X

X=Yes

4 Streaming

4.1 Overview

Not all sections in this chapter are actually about streaming, only related. This chapter spans over several issues concerning streaming: definition, data retrieving, alternatives to streaming, transport layer protocols and technologies that potentially can be used for streaming SOAP.

The first section defines how the terms "streaming" and "chunking" is used in this thesis. Then there is a section on strategies on retrieving data from a database. The two sections after that are related to SOAP. One is about a transport layer protocols and an application layer protocol that can be used to stream SOAP, and the second is about technologies that can be combined with SOAP and potentially provide streaming capability.

4.2 Defining the term "streaming" in this thesis

4.2.1 What we mean by "streaming"

A variety of definitions of the term "streaming" exists. The term is used in scenarios from water to electricity and data. Common for them is that something flows, or streams, from one end to another.

In the world of computers, streaming refers to the flow of data from one end to another. One of the everyday areas where streaming is commonly used, is streaming video and/or sound (e.g. watching the news over the Internet). It is referred to as streaming because the viewer can watch the start of the show while the rest is being downloaded.

Our definition of streaming, or what we refer to as streaming in this thesis, has the following criteria: the transfer must begin immediately without having to construct the data in memory first. Thus, our definition of streaming includes requirements for the server side, but not for the client side. Being able to process the beginning of the response before the entire response is received is not required.

If for example a SOAP server receives a request for large amounts of data, builds up the response in memory and then transfers it to the client, this is not considered streaming.

4.2.2 What we mean by "chunking"

In a sense, all transfer of data on the Internet is chunking. All data on the Internet are transferred in packages, so a constant stream of data is a constant stream of packages. And, therefore chunking.

Our definition of chunking, or what we refer to as chunking in this thesis, has the following criteria: making a conscious choice on the size data, and transferring it.

Collecting 10 lines from a database, putting them in an XML document and transferring them. Or collecting 1000 lines form a database, putting them in an XML document and transferring 1 KB of that document in one chunk.

Collecting 1000 lines from a database, putting them in an XML document and transferring them is not considered chunking, even if the underlying protocols transfers the document in chunks of 1 KB.

Here are some example scenarios where chunking data can be useful:

- If larger amounts then the protocol supports are to be transferred.
- In systems with limited resources or extensive usage, chunking can provide a way to use the resources more efficiently (e.g. chunking data to fit the transportation layer).
- With unknown amounts of data, chunking allows for more streamlined processing.

4.2.3 Defining XML streaming

The term "XML streaming" is in this thesis used to describe the process of streaming an XML document from a server side to a client side. It involves no criteria for processing the XML real-time, as it is received.

4.3 Streaming from database

4.3.1 Introduction

Retrieving large amount of data from a database, can strain the server resources unnecessary, especially if several clients are connected at the same time. It is therefore important to keep memory usage as low as possible. Streaming from the database reduces this problem.

This section covers two strategies to fetch data from a Structured Query Language (SQL) database, either by fetching all at once or by fetching one row at the time to create a stream of rows from the database.

4.3.2 Accessing data from database

One strategy when accessing data from a SQL database is to store the data in a dataset. The dataset is an in-memory cache of records that you can work with while disconnected from the database. This means that large amount of data is placed in memory.

Another strategy is to access the database directly to avoid this memory build up. The data is streamed from the database, and only a small amount is in memory at one time.

An example of how these strategies are used is in ADO.NET in the .NET Framework. The two central components in ADO.NET are the DataSet and the .NET data provider, see Figure 4.1. The DataSet and DataAdapter allow you to create a client side cache of a related record set. It is the DataAdapter who takes care of updates against the database.

The .NET Data Provider is design for data manipulation and forward only, read only access to data. Use of the DataReader provides access to this stream directly without the use of a DataSet.

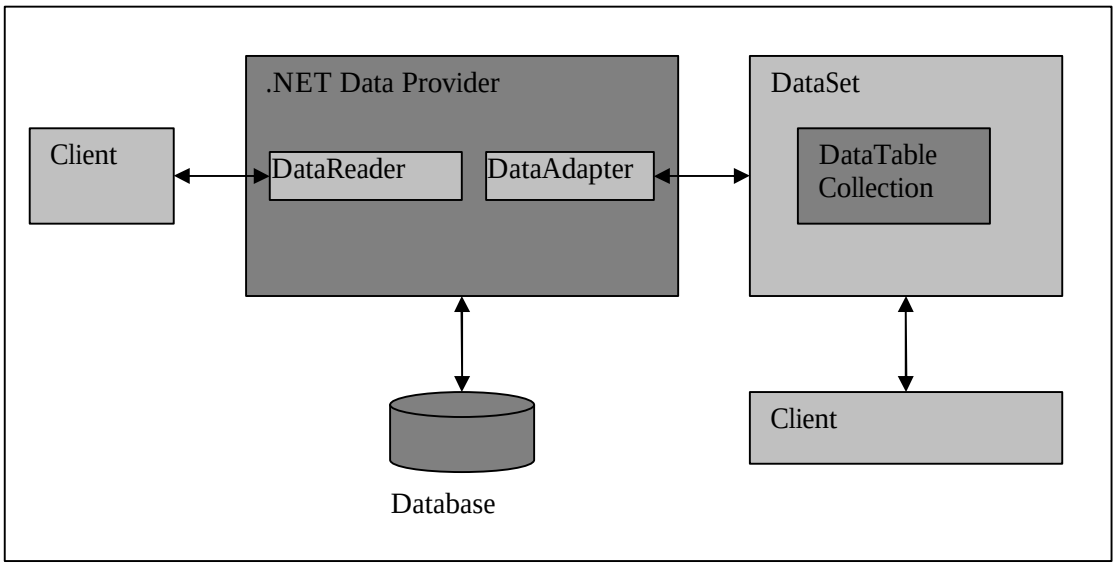


Figure 4.1 DataSet and DataReader overview.

The SqlDataReader provides a non buffered stream of data from a SQL database. This means that only one row is in memory at one time and this reduces system overhead and may increase application performance. Since the data is not cached in memory this method is a good choice when retrieving large amount of data from a database.

4.4 Protocols used for streaming

4.4.1 Transmission Control Protocol (TCP)

All hosts on the Internet are identified by an Internet Protocol (IP) address. And the most common way of communicating on top of IP is trough either TCP or User Datagram Protocol (UDP). These relate to the Open Systems Interconnection (OSI) model in the following way:

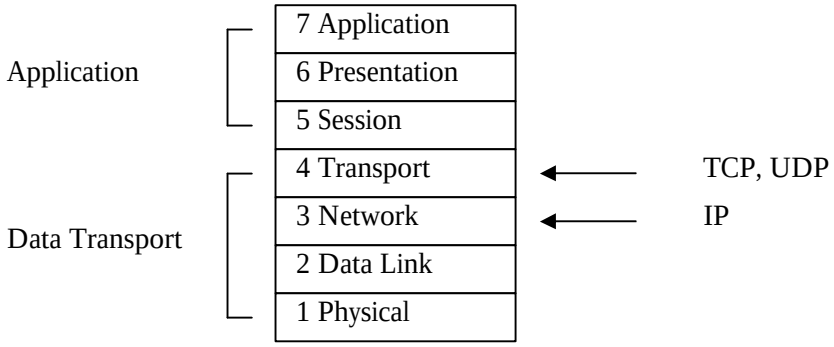


Figure 4.2 The OSI model

TCP/IP connections are identified by IP and port number for both peers. A port is number from 0 to 65535 which identify a TCP connection. The number relates to the size of the address field in TCP.

In general, UDP is used for communication where timing is essential and 0 % loss is not required (e.g. video streaming and online gaming). TCP is in general used when reliability and 0 % loss is essential. In this thesis, TCP is used as 100 % reliability is necessary.

For more information on this, please see TCP specification [41].

4.4.2 Hypertext Transfer Protocol (HTTP)

RFC 2616 "Hypertext Transfer Protocol -- HTTP/1.1" [23] defines the HTTP protocol.

HTTP is a client to server protocol with which two machines can communicate over a TCP/IP connection. A server listens on a specified port, usually 80, for HTTP requests. The client initiates the connection and sends a request, and the server then responds.

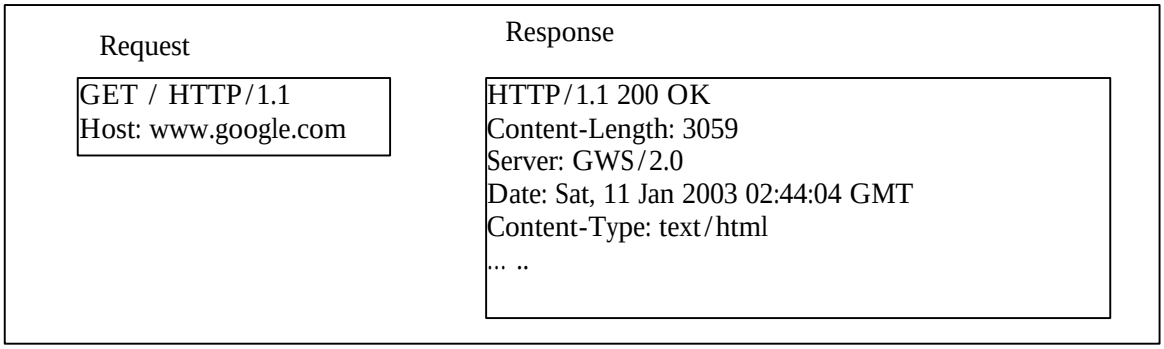


Figure 4.3 HTTP request/response example.

"GET" is used for retrieving information.

"200 OK" means the request was received, understood and accepted.

It is fully possible to stream the HTTP response. If for example the response includes data retrieved from one or more databases, the start of the message can be transferred before the rest is retrieved.

It is also possible to traditional video and sound streaming, not the kind of streaming we defined for this thesis, over HTTP. QuickTime from Apple does this by putting a new protocol on top of HTTP.

4.5 Alternative transfer technologies

4.5.1 Multipurpose Internet Mail Extensions (MIME)

MIME is a specification for formatting non-ASCII messages so they can be sent over the Internet. It was developed to widen the range of use for Email. MIME is now specified in RFCs 2045 [12], 2046 [13], 2047 [14], 2048 [15] and 2049 [16].

The material presented here is a short summary of the part of MIME that is relevant to this thesis, please refer to the RFCs for more details.

The MIME header has five fields defined: MIME-Version, Content-Type, Content-Transfer-Encoding, optional Content-ID and optional Content-Description.

```
From: Ørjan Stallemo <ostallem@siving.hia.no>
To: kvatne@siving.hia.no
Subject: mail test
MIME-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

This is the actual message
```

Figure 4.4 Example Email

A single MIME message is in Figure 4.4 used for the purpose of transferring an Email without any attachments. The optional headers Content-ID and Content-Description have been dropped.

```
From: Kristian Vatne < kvatne@siving.hia.no >
To: ostallem@siving.hia.no
Subject: mail test 2
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="*****"
--*****
Content-Type: text/enriched; charset="us-ascii"

This is the actual message
--*****
Content-type: audio/basic
Content-transfer-encoding: base64
Content-description: Some song

(Some song in an audio format)
--*****--
```

Figure 4.5 Multipart message example.

Multiple MIME messages are used to send an Email with "Some song" as attachment. Each MIME message is separated by a boundary, with "--" in front of them. "--" also appears at the end of the last message.

4.5.2 Direct Internet Message Encapsulation (DIME)

DIME is a lightweight binary message format for sending and receiving messages. Although SOAP is the primary reason for creating DIME, it is not required that DIME is used with SOAP.

DIME has no current specification. The last was an Internet-Draft published by IETF called Direct Internet Message Encapsulation (DIME). It expired in December 2002. As of May 2003, a local copy can be attained from Microsoft [11].

Summary of the relevant aspects of DIME messages:

- ? DIME is a specification for including multiple binary records within a single package.
- ? DIME has no restriction on the size or format of the data: they may vary from record to record within the same message.
- ? DIME does not need to know the length of the total data before sending.
- ? DIME can utilize chunking to deal with large amount of data, or to better deal with unknown amounts of data.

As explained in section 4.2.2, chunking allows for data to be divided into smaller parts. This ability is native in DIME.

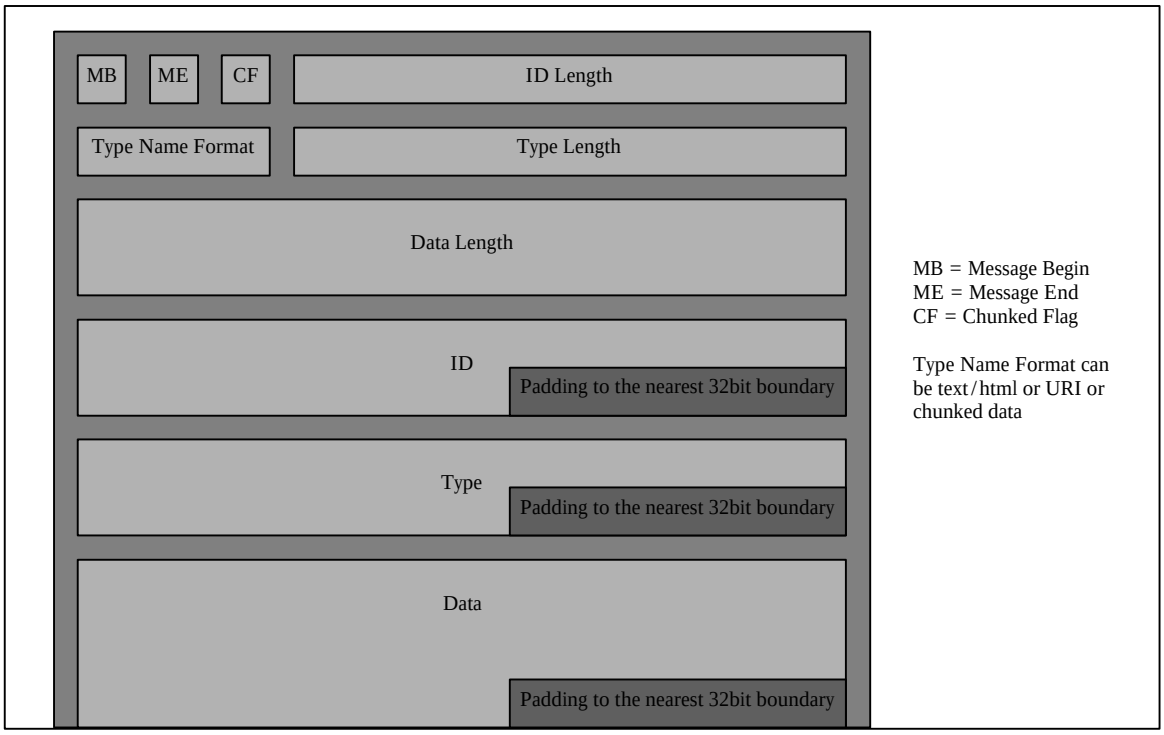


Figure 4.6 DIME Record Format.

Figure 4.6 gives an overview of the anatomy of a DIME record: it consists of predefined fields with constant lengths. The figure does not show all fields specified in DIME, only the relevant ones. Following that is a principle schematic of a DIME message in Figure 4.7.

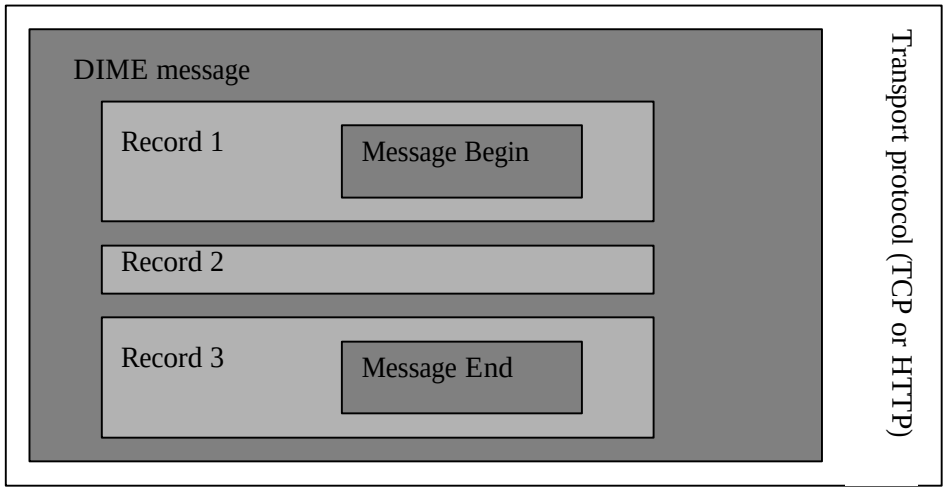


Figure 4.7 Principle schematic of a DIME message.

4.5.3 Blocks Extensible Exchange Protocol (BEEP)

4.5.3.1 Introduction

BEEP is considered a framework, and takes care of connection, authentication and packaging on top of the transport layer. Though it is not necessary to map it onto TCP, it is common.

The "Blocks Extensible Exchange Protocol Core" is specified in RFC 3080 [7]. RFC 3081 [8] defines how to map BEEP onto TCP. For more articles and projects on BEEP, please see beepcore.org [42].

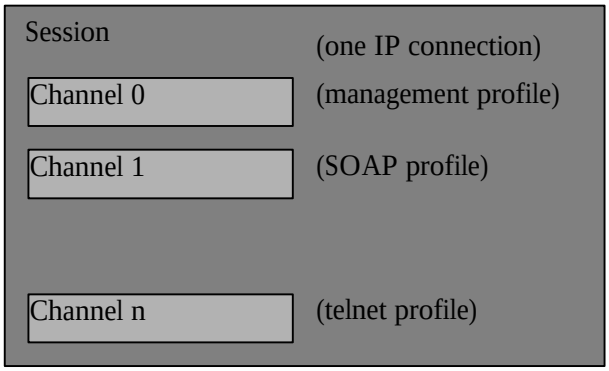


Figure 4.8 BEEP session principle.

BEEP is a peer-to-peer protocol, meaning there are no traditional client and server. One side sets up the connection, and requests for more channels. If the host at the other side supports the profile requested for, a channel may be set up.

Profiles define how messages are exchanged between the peers. This includes encryption, authentication and other exchange rules.

5 Feasibility study

5.1 Overview

This chapter is a feasibility study of strategies for overcoming the problem with excessive strain on server resources when transferring large XML objects over SOAP.

The first section outlines solutions to the problem that does not involve streaming, but looking at message patterns and alternative transport means, the next is about combining SOAP with the technologies described in section 4.5, and the last section looks at the why SOAP does not support streaming.

5.2 Alternatives to streaming SOAP

5.2.1 Sending a URI

Perhaps the simplest way of transferring a large XML document with SOAP would be to respond with an URI to where the serialized XML document can be retrieved, and a description of the transfer medium. The medium could be anything from HTTP, File Transfer Protocol (FTP) or even a TCP socket.

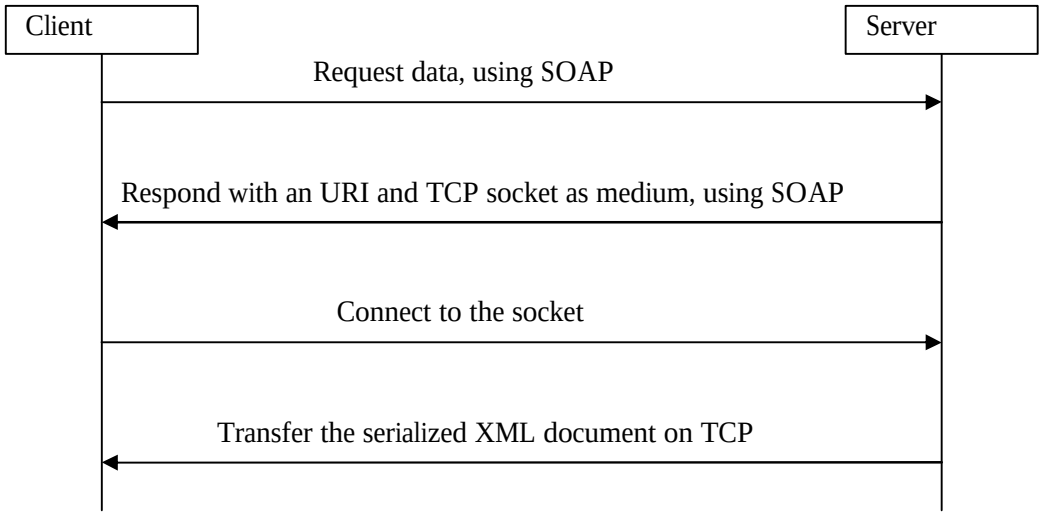


Figure 5.1 Sequence chart for sending an URI and medium type with SOAP.

This approach solves the memory problem on the server side, but it does not really use SOAP the way it was intended in this thesis. It sidesteps the use of SOAP on the difficult part entirely by only using SOAP to set up the connection, and using another medium to transfer the data.

5.2.2 Single request, multiple responses

Section 2.4.2 states that SOAP does not necessarily have to be transported on top of HTTP, which is limited to a request/response message pattern. A possible solution to the problem of transferring large XML documents could be to use a Request/N-Response pattern, and divide the data over several SOAP message.

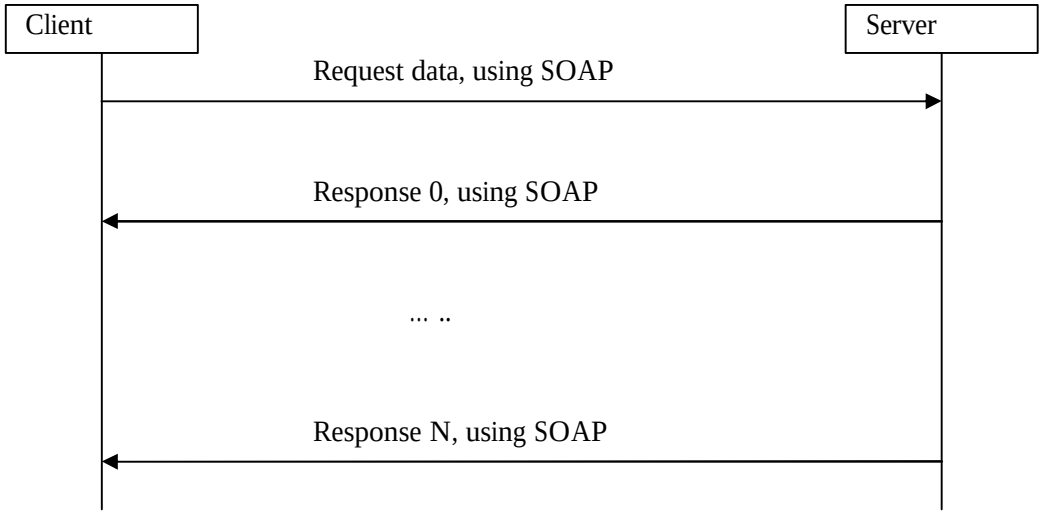


Figure 5.2 Sequence chart for Request/N-Response communication with SOAP.

The XML document parts could either be sent as part of the primary SOAP message, or as an attachment using for example MIME or DIME. See section 5.3.1 and 5.3.2 for details.

As HTTP does not support this type of message pattern, another transfer protocol must be used. Potential protocols for this sort of message pattern are DIME and BEEP. These are described in use with SOAP in section 5.3.2 and 5.3.3.

5.2.3 Multiple requests, multiple responses

Sending multiple requests and receiving multiple responses, or an N-Request/N-Response message pattern, is also a possible way of overcoming the streaming problem.

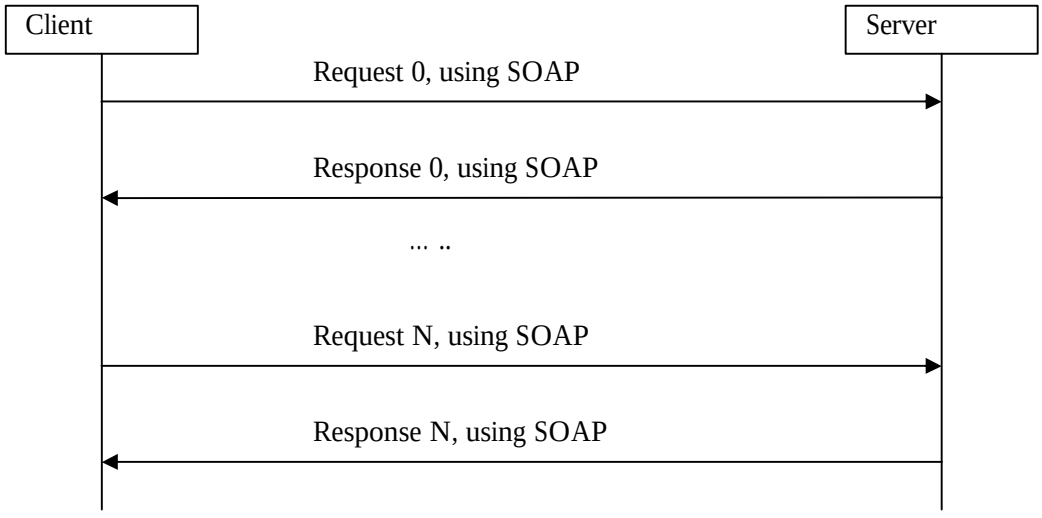


Figure 5.3 Sequence chart for N-Request/N-Response communication with SOAP.

In a message pattern like this, normal SOAP semantics can be used. There is no need for additional technologies to overcome problems with large amounts of data.

5.3 Combining SOAP with alternative technologies

5.3.1 SOAP Messages with Attachments (SwA)

SwA defines how a SOAP message can be carried in a MIME multipart/related message, and still abide the rules for SOAP messaging. This section sums up the aspects that are relevant to this thesis. The specification is a submission to W3C called "SOAP Messages with Attachments" [10].

MIME Multipart/Related (RFC 2387 [17]) remains semantically similar to a SOAP protocol binding so the SOAP message is unaware that it is being capsulated. SOAP processors receiving a MIME Multipart/Related with a SOAP message in the root part will process the message according to the SOAP specification [3].

A multipart format named "SOAP Message Package" has been defined. These are constructed using the Multipart/Related media type, and follows these rules:

- The primary SOAP message is carried in the root body part of the multipart/related structure.
- MIME attachments are referenced inside the SOAP message using SOAP references.
- Referenced MIME attachments must contain either a matching Content-ID header or a matching Content-Location header.

SOAP References to Attachments specifies how the primary SOAP message must refer to other entities in the message package. This is done by using existing mechanisms in SOAP and MIME.

The HTTP binding for SwA describes the relationship between HTTP headers and the MIME headers, not whether or not an asynchronous messaging or a synchronous request/response interaction pattern should be used.

5.3.2 SOAP and DIME

This section describes the relevant rules for how to encapsulate SOAP in DIME. Please see section 4.5.2 for more details on DIME, including specification.

In the article "Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation" [21], Jeannine Hall Gailey lists a number of things to keep in mind when using DIME with SOAP:

- The first DIME record contains the primary SOAP message. Additional SOAP attachments are included in subsequent record payloads.
- The primary SOAP message may cross-reference any subsequent attachments by a Universal Unique Identifier (UUID) or any form of URI. This UUID is indicated by the href attribute and is used to match the ID field of the corresponding DIME record.
- When cross-refer-encing attachments, relative URI references should be converted to absolute URI references.
- When binding DIME messages to HTTP, the HTTP Content-type header field must specify "application/dime" instead of the usual "application/soap+xml" or "text/xml" defined respectively by the SOAP 1.2 and SOAP 1.1 protocols.
- For DIME attachments that are signed or encrypted, security information about the attachment should be included in the header of the primary SOAP message.

Figure 5.4 shows a DIME message encapsulation a SOAP message and attachments. The DIME message can be sent either on top of TCP or on HTTP.

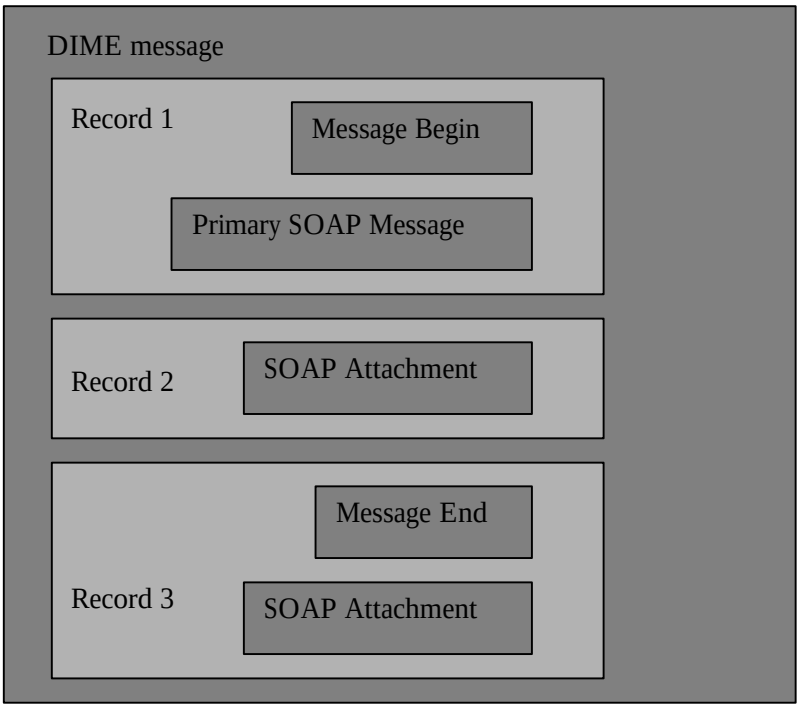


Figure 5.4 SOAP encapsulated in a DIME message.

5.3.3 SOAP and BEEP

RFC 3288 "Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)" [9], specifies how to use SOAP with BEEP.

From section 4.5.2 we know that BEEP sits on top of the transport layer, much like DIME. This relates nicely to what is described in section 2.4.2: that SOAP is most commonly but not necessarily used with HTTP.

The BEEP profile for SOAP is identified as "http://clipcode.org/beep/soap" [43]. The side of the peer that acts like a server identifies the "virtual host" through the serverName attribute. So the request to start a SOAP channel could look like the message in Figure 5.5.

```
<start number='1' serverName='stockquoteserver.com'>
        <profile uri='http://clipcode.org/beep/soap' />
                <![CDATA[<bootmsg resource='/StockQuote' />]]>
        </profile>
</start>
```

Figure 5.5 Message to start a SOAP profile channel in BEEP example.

Furthermore, the BEEP profile for SOAP defines the use of envelopes encoded as UTF -8 using the media type application/xml. And in case of attachments, MIME is used.

5.4 The underlying problem

5.4.1 Why wont SOAP stream

From section 2.4.2, we know that the most common way of transferring SOAP message is on top of HTTP. We start by looking at streaming SOAP over HTTP.

We have the following two facts:

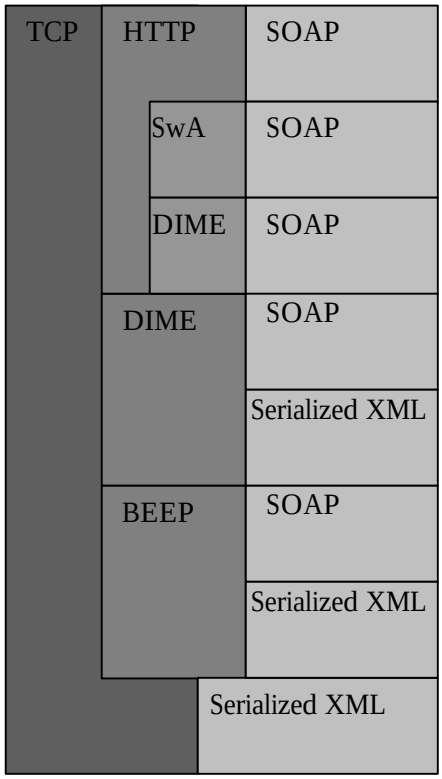
- A HTTP response with fault code 2XX (e.g. 200) indicates that a request was successfully received, understood and accepted.
- SOAP specification, see section 24.4, states that a server must issue a HTTP fault code "500 Internal Server Error" if an error occurs during the processing of a request.

This means that while it is possible to start streaming the start of the response while the server is still processing the rest of the response, the SOAP specification does not allow this.

It is of course entirely possible to simply ignore this issue, and make an implementation that replies "200 OK" in accordance to HTTP but not to SOAP. We have not been able to locate any implementation that does this, but the issue has been raised by Andy Neilson on the W3C's discussion boards [22].

The HTTP binding is the only binding described in the SOAP specification, so this limitation only applies to HTTP. If bindings to another transfer protocol could be specified and agreed upon, there is no theoretical hindrance to achieving streaming with SOAP.

5.4.2 Protocol overview



Stream?	Request/N-Response?
No No support in SOAP	No No support in HTTP
No No support in SOAP	No No support in HTTP
No No support in SOAP	No No support in HTTP
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes

Figure 5.6 Protocol layers and supported technologies overview.

Figure 5.6 shows an overview of some of the possible protocols SOAP can be transferred on top of, and whether or not they support streaming or Request/N-Response. This overview is purely theoretical, so implementations many not support streaming or Request/N-Response even if it is possible.

6 Applications and adaptations for testing

6.1 Overview

This chapter is intended to provide information about the testing and implementation in this thesis, so others can reproduce our results. It also serves a way of showing the adaptations made when going from theory to testing.

The first part gives a short introduction to the software solutions we used. After that is a part on the software we wrote in other to conduct the testing in the next chapter. The last part is about the XML format and database we have used for testing, and some limitations in the development environment.

6.2 Software solutions

6.2.1 Visual Studio .NET

The decision to use Visual Studio .NET as the development environment in this thesis was made based on what Sense Technology used. The decision was not forced on us, but we felt it natural to adopt the development tool Sense Technology used.

During the time spent on this thesis, Microsoft has launched the new Visual Studio .NET 2003. But, in the starting phase of the thesis, Visual Studio .NET was the newest addition to Microsoft's developer platform.

Visual Studio .NET was the first development environment from Microsoft build up around the use of XML Web services. In this aspect it was a major improvement to its predecessor, Visual Studio 6.0.

For product information on Visual Studio .NET, see the product page [44].

6.2.2 XML Web Services downloads

6.2.2.1 Introduction

A handful of XML Web Services downloads are available for Visual Studio .NET. The purposes of these are to add features, help and validation functionality.

For this thesis, we installed SOAP Toolkit 3.0 and Web Services Enhancements (WSE) 1.0 Service Pack 1.

6.2.2.2 SOAP Toolkit

The SOAP Toolkit was developed to allow a programmer to add XML Web Service functionality to existing Component Object Model (COM) applications and components. The current release is version 3.

SOAP Toolkit 3.0 also includes a "Trace Utility" application, used to trace SOAP messages sent over HTTP. SOAP messages are sent through this application, and then passed on towards the final destination.

6.2.2.3 Web Services Enhancements (WSE)

WSE extends the functionality of web services for Visual Studio .NET and the .NET framework. This includes security, scalability, performance enhancements and more. Features that gain wide acceptance will be absorbed into the .NET framework.

Functionality in WSE 1.0 is based on WS-Security, WS-Routing, WS-Attachments, and DIME specifications.

The current release is WSE 1.0 Service Pack 1.

6.2.3 Microsoft SQL Server™ 2000

We have chosen to use Microsoft SQL Server 2000 because the Microsoft Windows ActiveX prototype implementation of the WITSML API 1.1.0A uses this server. WITSML API 1.2, who

is the current version, does not have a prototype implementation but since we are working on the windows platform we found it suitable to use the SQL server from Microsoft.

6.2.4 MemBoost

In order to conduct fair testing, all tests preformed had approximately the same available free memory. This was done by clearing the system for all memory not currently in use.

The MemBoost application by Dhruv Matani takes care of this. For more information, see the product page [45].

6.2.5 Windows XP Performance Monitor

We needed a tool to measure the performance of the computer while our tests were running. Since we were working in a Windows XP environment we decided to use Performance Monitor (Perfmon). The Performance Monitor is Microsoft's primary tool for measuring and monitoring system performance.

The tool supports automatic collection of performance data from a local or remote computer. This data can be viewed using System Monitor, or exported for further analyzes.

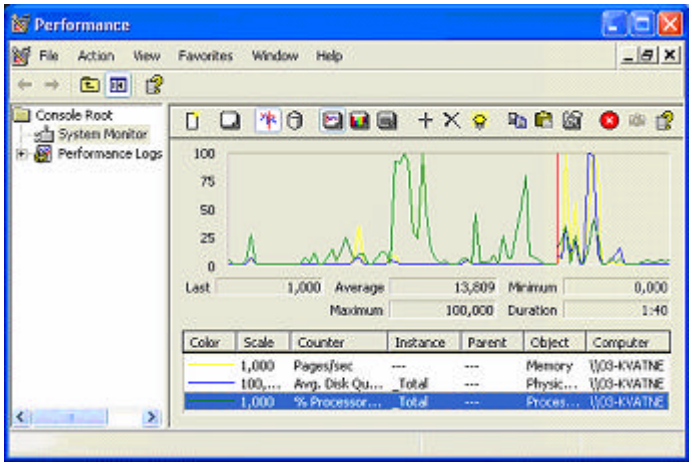


Figure 6.1 Perfmon application screenshot.

6.3 Test applications

6.3.1 Introduction

This section describes the applications that were made in order to conduct the tests presented in chapter 7. All applications are programmed in Visual Studio.NET with the programming language C#. Screenshots and the source code for each application are presented in Appendix B.

6.3.2 Compression tester

The purpose of Compression tester is to make the testing of the compression applications more dynamic and easier to conduct.

- Graphical User Interface (GUI) that incorporates the "command line" compression applications.
- Measures the time it takes to compress the files, and outputs in .txt files.
- Can do multiple compressions.

6.3.3 Generator

Generator is used to fill the database "XMLLog" with test data. Each time the program is executed, about 7500 rows are added. The format of these rows is explained in section 6.4.2.

6.3.4 Database stream tester

Database stream tester was made to test strategies data retrieving from database. The application has a GUI interface and the main features are:

- Possible to choose between two methods to fetch data, SqlDataReader that streams data and DataSet that reads all at once.
- Writes the database data in a file for verification.

6.3.5 SOAP N-Request tester

The SOAP N-Request tester has both a client and a server application. The purpose for this program is to test SOAP over a network. Another aspect is to test if SOAP performs better if a large data request is divided into multiple Request/Responses.

The server side is a WebService that returns a DataSet with the requested data.

The client side of the application has a GUI interface and the main features are:

- Can choose the interval of rows to fetch, and how many rows fetch in one call.
- Outputs XML object to file for verification.

6.3.6 DIME tester

The DIME tester application also has a server and a client. The purpose of this application is to test DIME streaming from a server to a client over TCP.

The features on the server side are:

- ? GUI to display debug information.
- ? Streams response back to the client as a DIME message divided into chunks of 4096 byte.

The client side has a GUI interface and the following features:

- ? Can choose the interval of rows to fetch.
- ? Outputs the data in file for verification

6.4 Adaptations to this thesis

6.4.1 XML structure

We constructed our own XML document format. It resembles some of the WITSML structures in that it has several elements and a few of them have large amounts of content. We felt that testing the principles were important, and not necessarily actual WITSML.

From Figure 6.2 we see that every "log" element has three children: ID, TIMEDATE and LOGDATA. The LOGDATA consist of 50 comma separated numbers of 5 digits and 6 decimals digits.

```
<xml>
  <Log>
    <ID>1</ID>
    <TIMEDATE>01.01.2003 01:00:00</TIMEDATE>
    <LOGDATA>91001.492425,... ..,50709.751110</LOGDATA>
  </Log>
  <log>
    <ID>2</ID>
    <TIMEDATE>01.01.2003 02:00:00</TIMEDATE>
    <LOGDATA>45671.4598425,... ..,24729.349870</LOGDATA>
  </Log>
</xml>
```

Figure 6.2 Self made XML format example.

6.4.2 Database XMLLog

Based on the XML format we decided on, we constructed a database as shown in Table 6.6. And we then used the Generator application to fill the database with 15000 rows of test data.

Table 6.1 Columns in table "log" from XMLLog database.

Name	DataType	Size	Nulls	Id
ID	int	4		X
TIMEDATE	datetime	8	X	
LOGDATA	text	16	X	

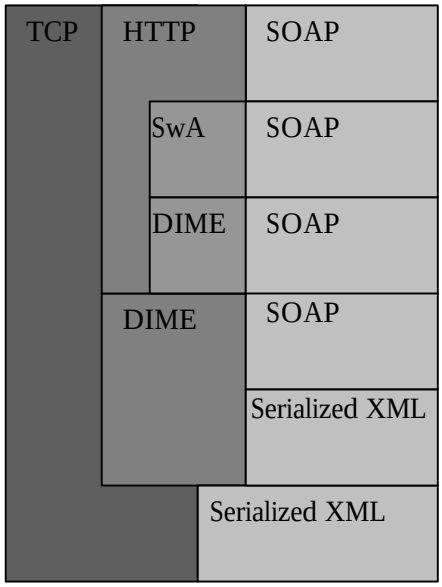
6.4.3 Limitations in Visual Studio .NET

Figure 5.6 in section 5.4.2 shows an overview of the theoretical possibilities for supporting streaming and Request/N-Response for different transport protocols. As we mentioned, this did not necessarily mean they where implemented. This is the case for Microsoft Visual Studio .NET.

ASP.NET WebMethods does not support streaming. Furthermore, every WebMethod ends with a "return" call, similar to a standard HTTP model. This excludes the Request/N-Response message pattern.

But it is possible to chunk a response back to the client. The problem is: SOAP extensions are disabled for the web service method when this option is utilized. Since the Sitecom System uses SOAP, this option could not be used.

An overview of streaming and Request/N-Response features supported in Visual Studio.NET, similar to Figure 5.6 in section 5.4.2, is shown in Figure 6.9. Note that BEEP is not implemented at all.



Stream?	Request/N-Response?
No No support in SOAP	No No support in HTTP
No No support in SOAP	No No support in HTTP
No No support in SOAP	No No support in HTTP
No No support in VS.NET	No No support in VS.NET
Yes	Yes
Yes	Yes

Figure 6.3 Protocol layer overview and supported technologies in Visual Studio .NET.

7 Testing

7.1 Overview

This chapter gives a detailed description of the various tests conducted. The description includes objective of the test, assumptions, limitations and a testbed description.

The test conducted first was a compression test of various compressors, then a test on methods for extracting data from a database, and finally a test on requesting data over network.

The last test part proceed in the following way: First is a test to determine what number of rows to retrieve for making N-Request/N-Response message pattern most effective. This result is used in SOAP N-Request/N-Response for the duration of the testing. The second and third tests test server performance for SOAP N-Request/N-Response versus DIME, with multiple clients connecting. The forth measures time used for DIME, SOAP N-Request/N-Response and SOAP when fetching data from a varying number of rows in our database.

7.2 Compression testing

7.2.1 Test objectives

To test compression of XML documents we picked a handful of different techniques. These are represented by third party implementations, in form of .exe files.

The way we see it there are two important aspects to compression, compression ratio and time used. As for time used, both compression and decompression are important.

From reading other compression tests [46] in the references, we have concluded that the size of the data to be compressed is the key factor to affect the outcome. Some techniques may perform very well on small amounts of data, and bad on large amounts. A relatively wide range of document sizes will therefore be tested.

The techniques, data and amounts of data in our tests have been chosen based on what is most relevant to the Sitecom project.

We expect that no technique will be superior to all the others in all aspects. Hopefully one will stand out as logical choice and the overall best technique.

7.2.2 Assumptions and limitations

The composition of techniques used for testing has been chosen based on two criteria:

- A widest possible range of techniques usable in one single environment
- Possible future use in Sitecom

To ensure fair testing we wanted all tests to be conducted in the same environment, even if it meant excluding some of the techniques. The environment which gave us the widest range of techniques was "command line" tools for MS Windows. GNU/Linux and .NET components where also considered.

Section 3.3.6 describes the most powerful tool for compressing XML documents: DTD/Schema awareness. Although the benefits are great, we do not recommend the use of this technique. In an environment where streaming and chunking are possible solutions to overcome large documents, committing to a DTD or Schema is not recommended.

Most of the techniques have one or more parameter options to help increase the performance. Utilizing all these options would be too time-consuming. And utilizing for one or two of the techniques would make it an uneven test. Regrettably, we therefore had to disregard all options.

7.2.3 The techniques

Based on the criterions above, we chose the following two text compressors and two specialized XLM compressors to test:

- bzip2 [38].
- Deflate64 in form of PKZIP implemented by PKWARE [39].
- XMill [40].
- XMLPPM [6].

7.2.4 Testbed description

To test compression, we use the application "Compression tester", described in section 6.3.1. This application does not perform any compression, it calls other applications that does the actual compression, and measures the time.

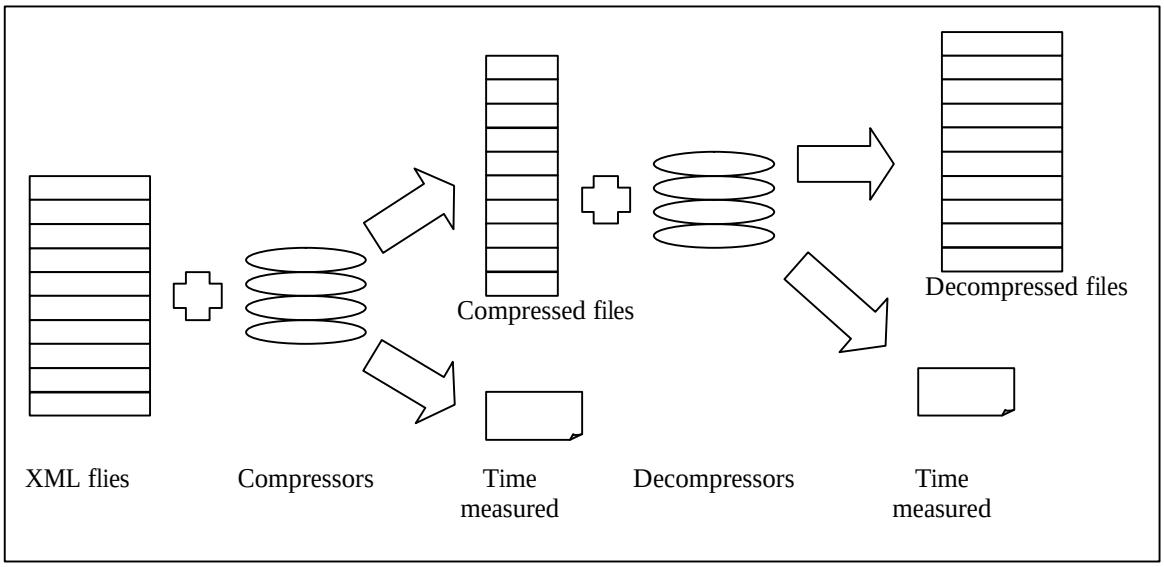


Figure 7.1 Compression tests overview.

The size of the XML document is recorded before compression, in compressed form and after decompression.

During testing, all processes not needed to complete the tests were shutdown, so all techniques would have the same CPU/memory available for the tests. Despite this, all tests were conducted 20 times, and an average was calculated. This was done to account for caching and unforeseen events that might use CPU power or memory.

7.3 Database streaming

7.3.1 Test objectives

To test if streaming from a database helps on reducing CPU and memory use on the server, we have picked two methods, SqlDataReader and DataSet in Visual Studio .NET, for testing. SqlDataReader stream one row at the time from the database and the DataSet fetch all at once.

The two tings we look for in this test are CPU and memory use. To ease strain on server, these must be as low as possible. This is just a simple test to see how these two methods differ, and we expect that SqlDataReader will be better both in CPU and memory use.

7.3.2 Assumptions and limitations

In these two test methods for fetching data from a database is used to see the difference between streaming and fetching multiple rows. We will not suggest that these are the two best methods, but we can suggest whether streaming has potential.

We chose to test these methods because they were integrated in the Visual Studio .NET environment and were easy to combine with our Microsoft SQL Server 2000.

7.3.3 Testbed description

To test streaming from a database, we use the application "Database stream tester", described in Section 6.3.4 and the measuring tool "Perfmon", described in Section 6.2.5.

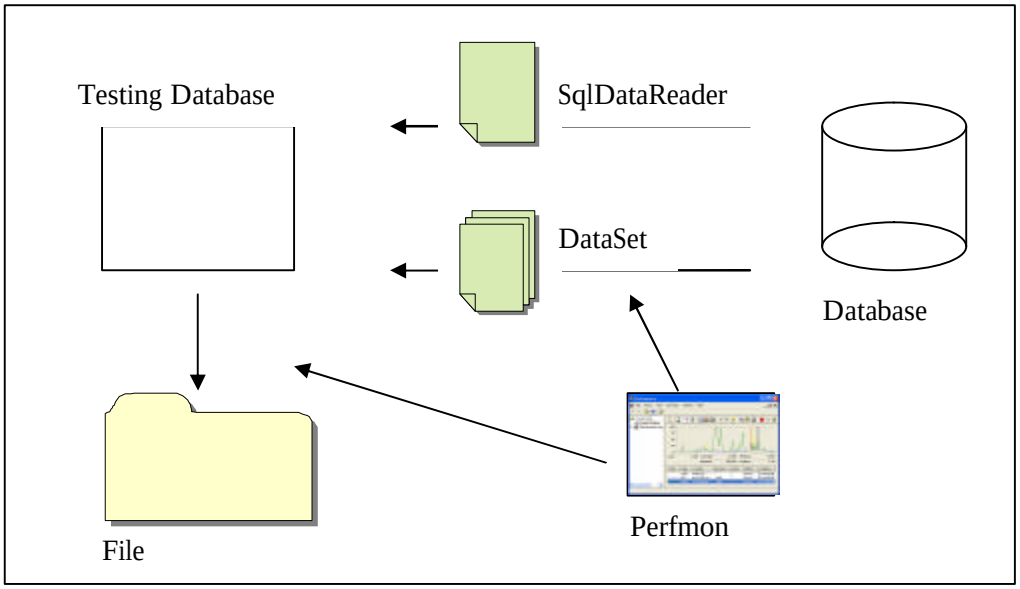


Figure 7.2 Database streaming test overview.

The application fetches the two columns "ID" and "LOGDATA" for all the rows in the database and then writes them to a file. Perfmon monitor CPU/memory usage during the testing.

During testing, all processes not needed to complete the tests were shutdown, so both methods would have the same CPU/memory available for the test.

7.4 Data request over network

7.4.1 Test objectives

To test streaming of data over a network, we have chosen a streaming technique and ordinary SOAP. They will be compared to see how much streaming helps to reduce strain on server.

To test streaming of data we chose to use DIME over the network and SqlDataReader from the database. And when SOAP and DataSet is used, the client can either fetch all the data at once or split it into multiple requests/responses.

The primary reason we chose DIME to test streaming is that it is a new technology that potentially can be used with SOAP for streaming. This it not the case in ASP.NET WebMethod at this time, but we can implement streaming data with DIME alone.

The three tings we looked for to find the best performance in these tests are CPU, memory use and time measured. To ease strain on server resources, CPU load and memory usage must be as low as possible. It is preferred that the process time on server is short. We also want to find the approximate amount of data to fetch in one SOAP request to achieve best performance.

We expect that streaming will be more CPU and memory efficient, but it will be interes ting to see how well SOAP performs, especially when we split the data fetch into multiple calls. We expect less strain on server resources, and increased processing time.

7.4.2 Assumptions and limitations

The methods for testing have been chosen on these criteri a:

- ? The method is possible to implement in Visual Studio.NET.
- ? One streaming method to compare with SOAP.
- ? One implementation of SOAP.

To ensure fair testing all the tests were conducted with the same available server resources. This was done by closing all processes on the server machine that was not necessary to conduct the tests. We also ran the memory manager tool MemBoost, see Section 6.2.4, between every test to ensure that approximately the same amount of memory was available for each test.

7.4.3 Testbed description

7.4.3.1 Introduction

To test the SOAP and DIME streaming method we used the two applications "SOAP N-Request tester" described in section 6.3.5 and "DIME tester" described in section 6.3.6. To measure the performance we used Perfmon, described in section 6.2.5.

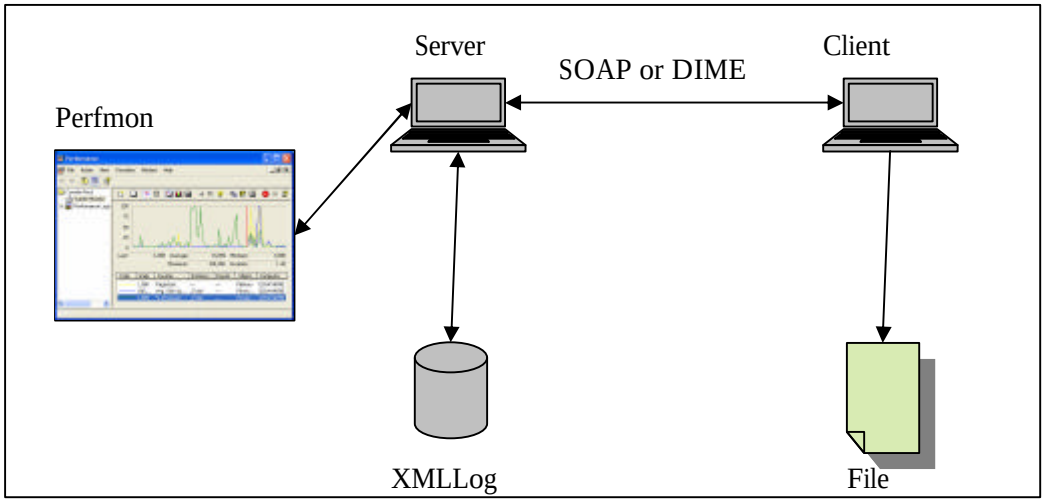


Figure 7.3 SOAP and DIME test overview.

We divided our test into four separate tests:

- ? SOAP N-Request.
- ? SOAP N-Request for 1-5 clients.
- ? DIME for 1-5 clients.
- ? Time test for SOAP, SOAP N-Request and DIME.

Perfmon logs counters on the server side during the testing. There are four counters and the logging occurs every 5 seconds. The counters are shown in Table 7.1.

Table 7.1 Performance counters.

Performance object:	Counters:
Memory	Available Bytes
Process	% Processor Time Working Set
Processor	% Processor Time

In the last test a different approach was chosen, as this test focus only on the time measured. To ensure a more accurate time the measuring tool on the server side sampled the time with an interval of one second. To compensate for the extra workload this puts on the server only the "Process/Processor time in percent" counter was used in this test.

7.4.3.2 SOAP N-Request

The first test we conducted was SOAP with multiple request/response to find what number of rows to fetch in one call.

The "SOAP N-Request tester" application client uses the web service on the server to fetch all rows in the database "XMLLog". SOAP is used in this transaction. The server uses DataSet to fetch data from "XMLLog". All the data is generated on the server before it is return to the client. The client can choose what number of rows to fetch in one call.

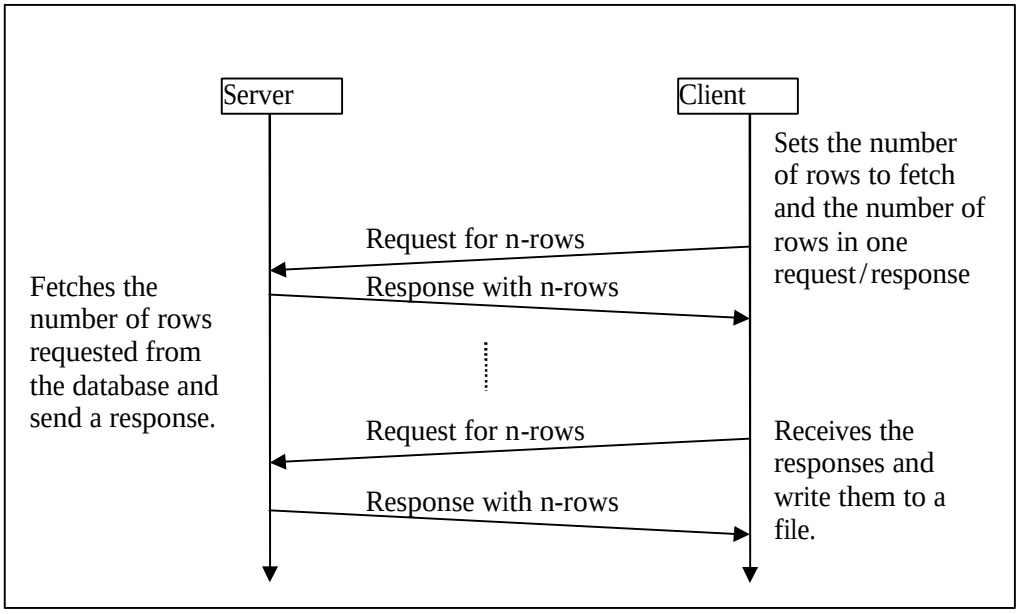


Figure 7.4 SOAP with N- Request/N-Response overview.

7.4.3.3 SOAP N-Request for 1-5 clients

This test was conducted to evaluate the performance of N-Request/N-Response calls using SOAP with multiple clients. From 1 to 5 clients was connected to the server at simultaneously. Based on a review of the results from the above test, see section 8.4.3, we decided to fetch 1000 rows in each call.

The "SOAP N-Request tester" application is also used for this test.

7.4.3.4 DIME for 1-5 clients

A similar test as the above was conducted on DIME streaming using the application "DIME tester". This was done to be able to compare the performance to SOAP vs. streaming DIME. From 1 to 5 clients was connected at the same time.

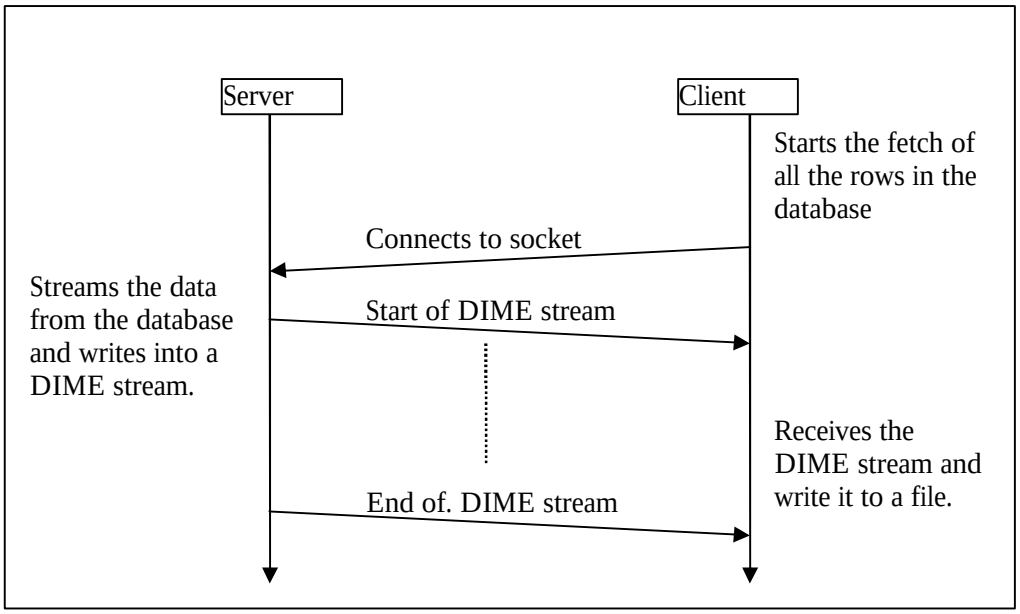


Figure 7.5 DIME streaming test overview.

The application "DIME tester" client connects to a TCP socket on the server and use DIME to fetch the data from the database. On the server side the application uses SqlDataReader to create a stream of data from "XMLLog". This stream is wrapped in XML tags, and transferred via DIME messages split into chunks of size 4096 byte. Chunks are generated dynamically as the data is retrieved from the database and then streamed continuously back to the client.

7.4.3.5 Time test for SOAP, SOAP N-Request and DIME

The time test was done with the same applications as the other tests. The time the server application used to process the request from the client was measured when retrieving 1, 1000, 5000, 10000 and 15000 rows from the server. This was done with ordinary SOAP, SOAP N-Request with 1000 rows in each request and DIME.

8 Test results

8.1 Overview

Presented in this chapter are the results from the test conducted. The results are in form of graphs, and the actual numbers can be found in the appendices.

The results are presented in the same sequence as the test: compression, data retrieving from database and finally data request over network. The parts include preliminary testing and actual results. Also an introduction and post testing where that is necessary.

8.2 Compression test results

8.2.1 Introduction

The results from time measurements presented here are the average of 20 similar tests. All results from the compression tests can be found in Appendix C.

No variation was found in the size of the compressed and decompressed files when multiple tests were conducted on the same file with the same technique. This was true for all techniques.

8.2.2 Preliminary testing

8.2.2.1 Introduction

Preliminary testing was done to make sure the compression worked as we expected it would. It came as a natural part of setting up the "Compression tester" application, described in section 6.3.2. These tests gave us some rather unexpected results: XMill and XMLPPM had difficulties compressing some of the XML documents.

8.2.2.2 XML validation in XMill

XMill performs a XML validation as it compresses the documents. It is our belief that this validation does not work as intended. Here is an example of an error:

```
<continent id='f0_124'
  name='Australia/Oceania'/>
```

Figure 8.1 Section of the XML document that XMill reported error on.

```
Parse error in line 10:
Symbol '>' expected after '/' tag!
```

Figure 8.2 XMill's error message.

XMill reports error on the '/' between Australia and Oceania. According to the XML specification [5], there are no rules against using this character in attribute values.

8.2.2.3 ISO Latin-1 character set with XMLPPM

Another bug, or rather a lack of feature, was found in XMLPPM. American National Standards Institute (ANSI) character set spans over 256 characters. 0-127 are the same as ASCII character set, and 128-255 are similar to ISO Latin-1 character set. XMLPPM would not compress documents containing characters 128-255 from the ANSI character set (e.g. 'æ', 'ü' and '®').

8.2.2.4 Conclusion

Despite the obvious faults to XMill and XMLPPM, we still decided to go along with the testing as planed. We felt that the performance testing of the XML compressors where the key issue. And that the results would not be compromised by adapting the documents. Before a compression standard could be used in Sitecom, it would have to be tested thoroughly. Faults like these can be corrected.

So the following adaptations were made:

? ‘/ ’character in the attribute name or value where removed.
? ISO Latin-1 character set characters have been removed.

8.2.3 Test files

The XML used for testing come from 2 categories. The first is WITSML API example files, and the second is test files generated from our database. The four smallest come from WITSML, and the six largest we generated. All files resemble files that are used in the Sitecom system.

The files were selected and generated exclusively based on theirs size. We wanted to have spread variety of different sized files.

Table 8.1 List of the files used for compression testing.

XML document:	original size (byte):
capClient	405
wellbore_no_xsl	2642
mudLog_no_xsl	7392
rig_no_xsl	14782
SOAPtest0	34738
SOAPtest1	101838
SOAPtest2	505743
SOAPtest3	1004172
SOAPtest4	2001975
SOAPtest5	2995965

8.2.4 Compression results

Figure 8.3 shows the size in percentage of the original files. XMLPPM start out best on smaller files, and XMill and PKZip does best on larger files.

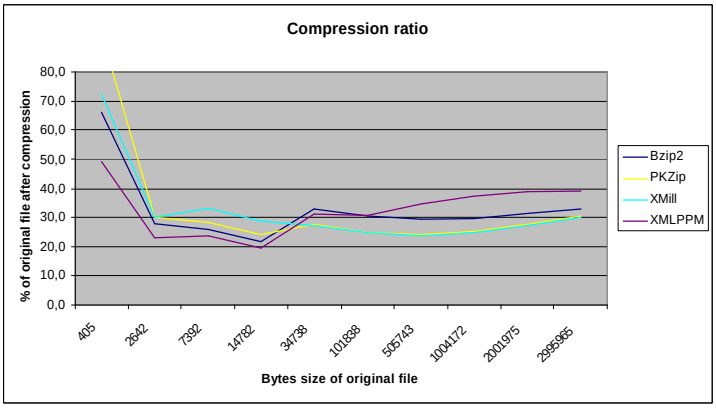


Figure 8.3 Compression ratio versus file size.

Figure 8.4 shows time measured for compression on a logarithmic scale. bzip2 does best on smaller files, and XMill on larger files.

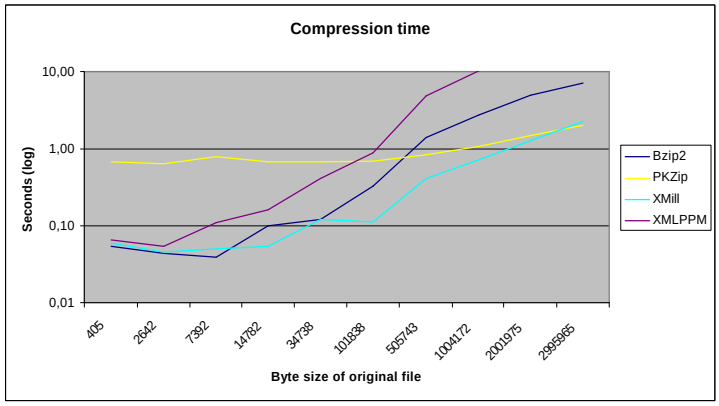


Figure 8.4 Compression time versus file size.

8.2.5 Decompression results

Figure 8.5 shows the size of the decompressed files in percentage, compared to the original file. XMLPMM does not reproduce the same files as the original.

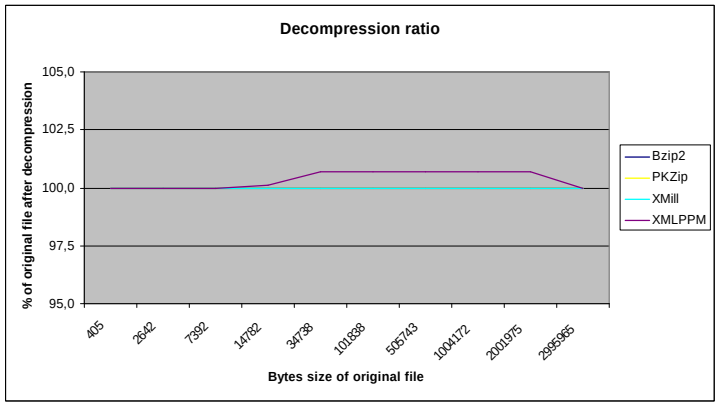


Figure 8.5 Decompression ratio versus file size.

Figure 8.6 shows the time measured for decompression on a logarithmic scale. The graph is very similar to Figure 8.4.

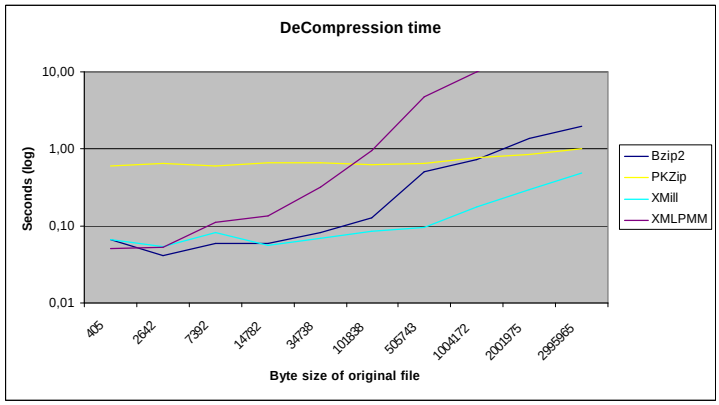


Figure 8.6 Decompression time versus file size.

8.2.6 Post testing

The test results from decompression were not quite what we expected. Further examination was needed to give us the answers to the deviance between the original and the decompressed document for XMLPPM.

XMLPPM outputs documents with code set "ANSI" and file type "PC". All of the original files we use for testing had code set "ANSI" and file type "UNIX". The difference between t hem is the way linefeed is coded. This cause a 0.7 % deviation is size.

Furthermore, we see that PKZip never compress in less than 0.6 seconds, while the others compress the smaller files in 0.04-0.06 seconds. We suspect that this is related to the fact that PKZip checks for license key every time it executes. Preventing this was not possible.

8.3 Database streaming results

8.3.1 Preliminary testing

Preliminary testing was done while the program was created. This was done to make sure that all worked in accordance with the test scenario.

One thing we found out was that the data is cached in memory when using dataset. This means that when we had done one test against the server we had to restart the client. If we did not the server would just use the dataset cached in memory to return the data.

8.3.2 Results

Figure 8.7 shows the applications CPU use on server. The values rise when the strain on server increases. SqlDataReader does not exceed 40 % while DataSet reaches 100 % in a short period.

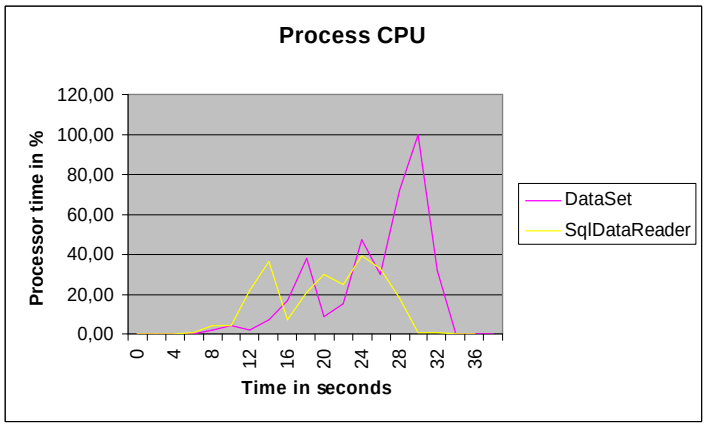


Figure 8.7 The process processor usage.

Figure 8.8 shows total CPU usage on server. SqlDataReader reaches 60 % and DataSet reaches 100 %.

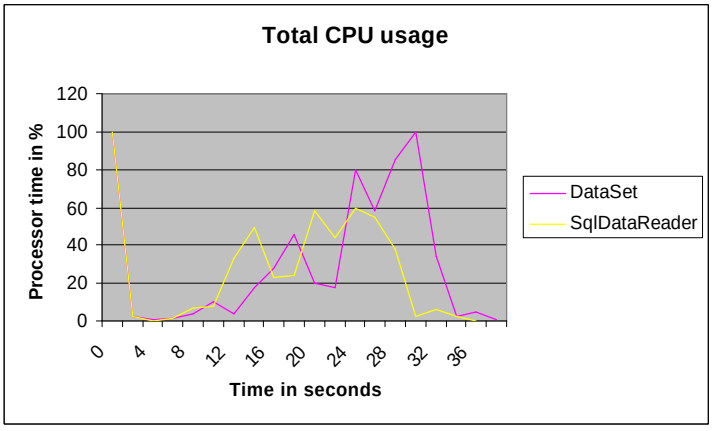


Figure 8.8 The total CPU usage.

To illustrate memory use, Figure 8.9 shows the total available memory during the test. The cure drops when the application uses memory.

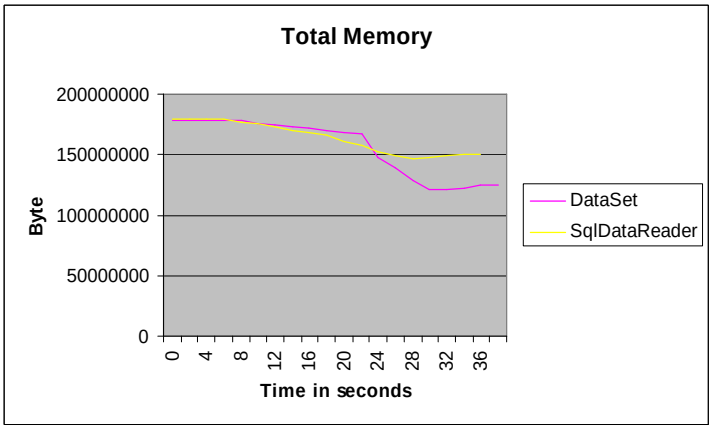


Figure 8.9 Available bytes per second.

8.4 Data request over network results

8.4.1 Introduction

The results from the measurements performed by Perfmon are presented here. All the test values can be found in Appendix C.

The graphs of test results in sections 8.4.2, 8.4.3 and 8.4.4 have curve named "15000 rows", "5000 rows" etc. This represents how many rows are fetched in one call with "SOAP N-Request tester".

8.4.2 Preliminary testing

Preliminary testing was done during and after setting up the different applications. It was done to make sure that the applications worked according to our expectations.

The peak on CPU usage, at the start of each curve in the "SOAP N-Request" test, is caused by the WebService being compiled and made ready to processing. This happens when the client makes the first request to the server. And it does not recur with multiple requests since the web service are already up and running when new requests are made.

When testing several clients simultaneously against the server using the DIME tester application, it timed out. The SqlDataReader SqlCommand generated the timeout and to compensate for this we expanded the timeout from 30 to 120 seconds.

As seen in Figure 8.10, the level of available memory at the start of each test varies. For example 5000 rows start with over 80 MB free memory, and 15000 starts with less than 60 MB. This was solved by restarting each application between every test and run the memory manager MemBoost.

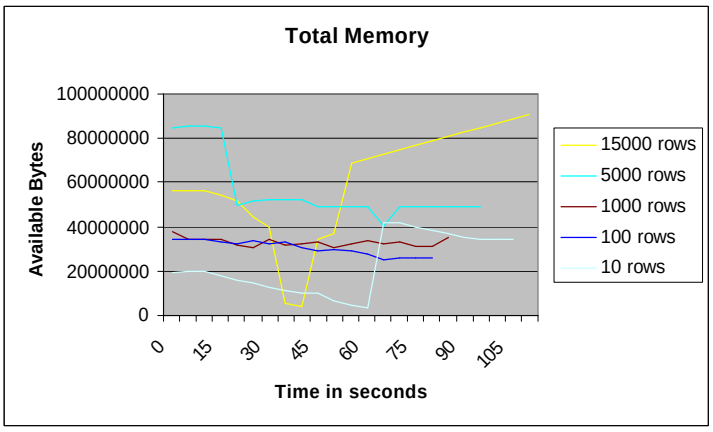


Figure 8.10 Available bytes in SOAP tests.

8.4.3 SOAP N-Request test

The client requests all rows in the database from the server. The number of call to the server depends on how many rows are fetched each call. If 100 rows are fetched in one call, the client must make 150 calls to get all the data since there are 15000 rows all in all.

Figure 8.11 shows the applications CPU use on the server. The results show that CPU usage in percentage is higher when retrieving large amount of data in one call.

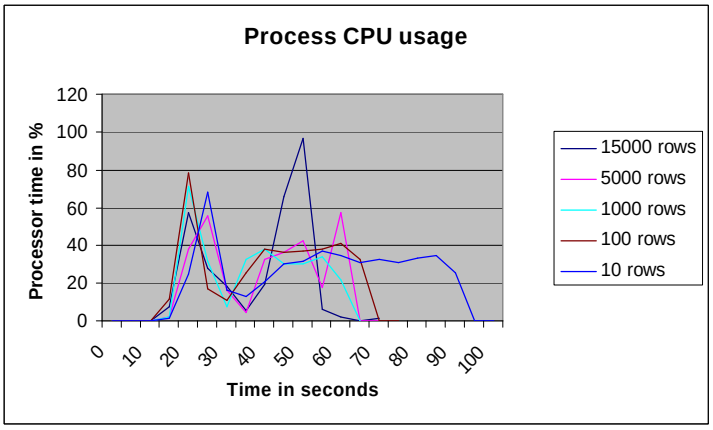


Figure 8.11 CPU usage for the web service "SOAP N-Request tester".

Figure 8.12 shows the applications memory use on server. The curves show the values of the process "Working Set" counter. Retrieving from 10 to 100 rows in one call performs almost similar with approximately 20 MB reserved in memory. 100 rows uses 22,4 MB, 1000 rows uses 32,7 MB, 5000 rows uses 71,5 MB and 15000 uses 130.6 MB.

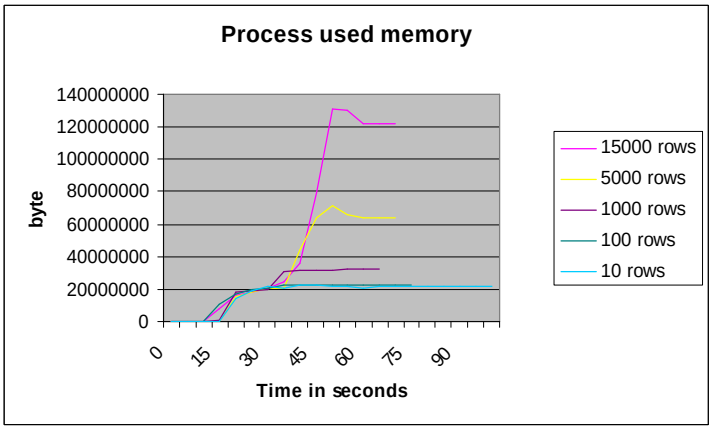


Figure 8.12 Memory reserved for the web service "SOAP N-Request tester".

To illustrate total memory use, Figure 8.13 shows the total available memory during the test. The cure drops when the application uses memory.

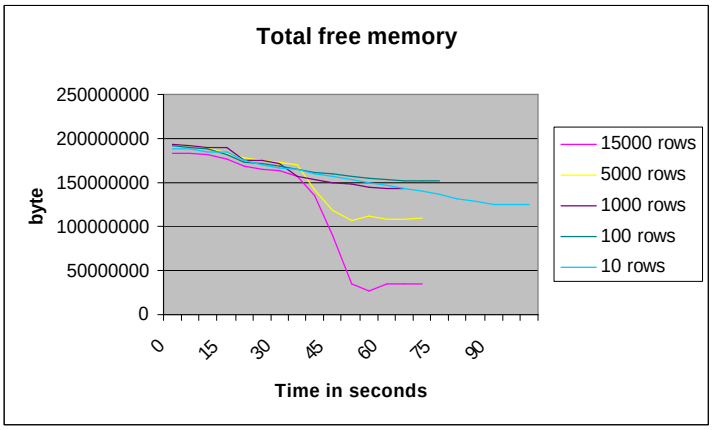


Figure 8.13 Total free memory on the server.

8.4.4 SOAP N-Request for 1-5 clients test

The client asks for the entire database and fetches 1000 rows for each call. Figure 8.14 shows CPU and Figure 8.15 shows process memory for 1-5 clients.

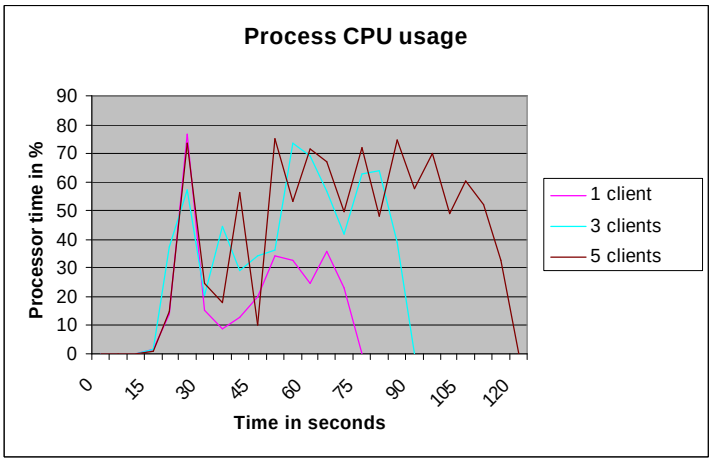


Figure 8.14 CPU usage for the web service "SOAP n-request tester".

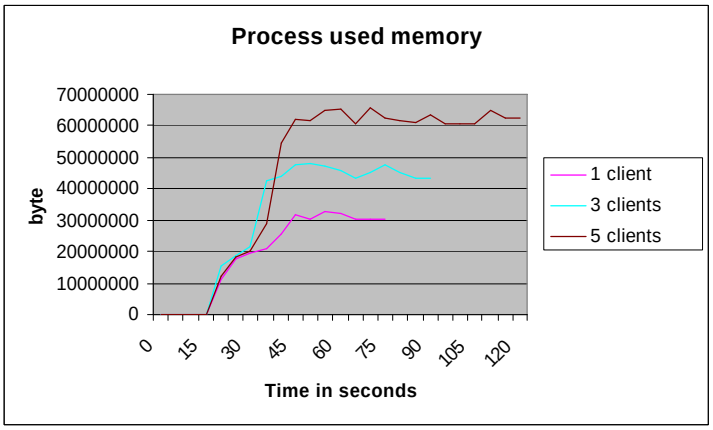


Figure 8.15 Memory reserved for the web service "SOAP n-request tester".

8.4.5 DIME for 1-5 clients test

The client gets all the data in the database "XMLLog". Figure 8.16 shows CPU and Figure 8.17 shows process memory use with different number of clients.

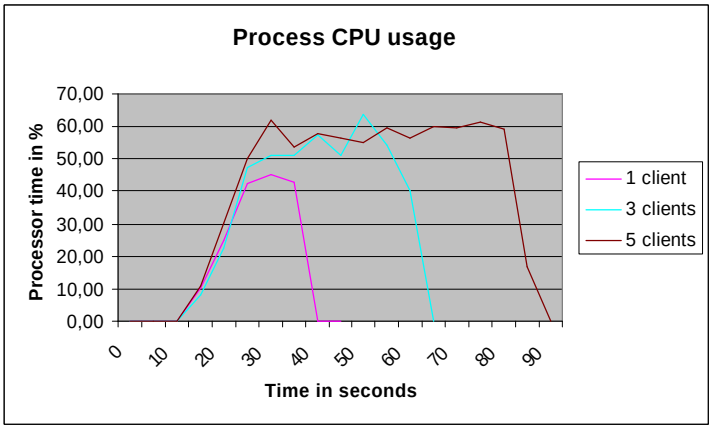


Figure 8.16 CPU usage for “DIME tester” application.

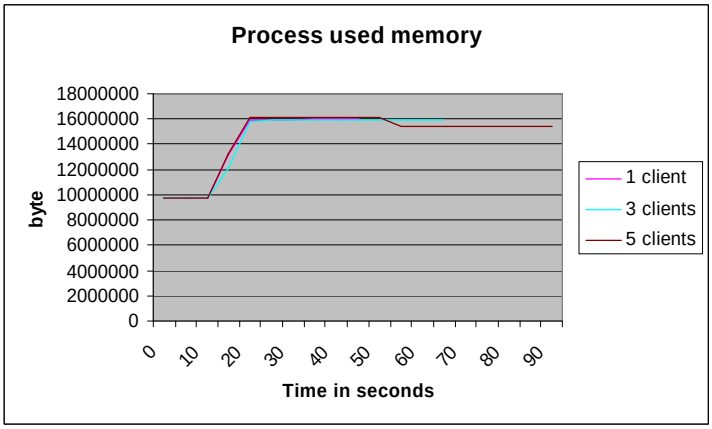


Figure 8.17 Memory reserved for the “DIME tester” application.

8.4.6 Time measure for server process

The test shows how much time the server used to process the clients request. Different number of rows is retrieved from the server and transmitted back to the client.

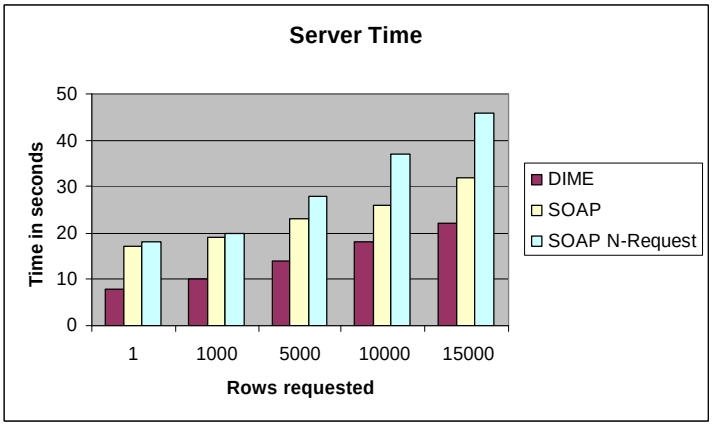


Figure 8.18 Time on server for different number of rows requested.

9 Discussion

9.1 Overview

In this chapter we discuss the test results from the last chapter, feasibility study from chapter 5 and the theory from chapters 2, 3 and 4. What we present here are our views and thoughts.

We first discuss compression, second is data retrieving from database, third is requesting data over network, and then we discuss the feasibility study. Last are some thoughts on future work on the subjects discussed in this thesis.

In our discussion, we consider only subjects directly at hand. Issues like security, implementation difficulties and costs are not considered as we have no basis for discussing it.

9.2 Compression

There is no clear winner in this test. But we see striking resemblance between the compression ratio of bzip2 and XMLPPM, and PKZip and XMill, especially on the larger files. XMLPPM performs best on the smaller files, and XMill and PKZip perform best on the larger files. bzip2 performs second best of the smaller files, and slightly worse than XMill on the larger. The point where XMill and PKZip outperform XMLPPM and bzip2 seem to be at about 20 KB. Test files closest to that point are 14 KB and 34 KB, so deciding on the exact size if difficult.

On time measured for compression, bzip2 performs best on the smaller files, while XMill performs best on the larger files. We also note that in part bzip2, but especially XMLPPM, performs badly on the larger files. XMLPPM performs nearly 18 times worse than PKZip on the 2.9 MB file. The results from decompression time are very similar to measurement on compression time, and no further interpretation is needed there.

As we mentioned in section 8.2.6, we suspect that PKZip is slowed down by a license key check. Evaluation its performance is therefore very difficult. We do however register that the time measured for compression is very constant for smaller files, and seem to climb at a lower rate then the rest of the techniques for lager files.

PKZip use the Deflate64 method, and XMill use Gzip for text compression, which use the Deflate method. See section 3.2.6 and 3.2.7 for a description. Comparing the compression ratios for theses techniques, we see that PKZip does a bit better at smaller files but they perform almost identical on files larger than 14 KB. We know that Deflate64 compresses better then Deflate, yet XMill compress almost equally good as PKZip. This indicates that specialized XML compression has some effect.

Furthermore we observe that our initial expectances were correct: there is no clear winner. While XMLPPM and bzip2 perform well on smaller files, XMill and PKZip do better at larger files.

For future usage in Sitecom we recommend the use of bzip2 for the following reasons: It is most likely that future versions will include either chunking or streaming feature, or both. In case of chunking, the sizes will probably not exceed 100 KB. In case of streaming, it is important that the compressor can handle pieces of a serialized XML document. The specialized XML compressors require an intact and complete document.

We do, however, recognize the potential in specialized XML compression, and recommend that Sense Technology keep them in mind.

9.3 Database streaming

SqlDataReader is the best method for minimize the CPU usage on server. We see that SqlDataReader does not exceed 40 % while DataSet reaches 100 % in a short period. When considering the total CPU usage on the server, SqlDataReader reaches 60 %. This is quite a high number, but it still outperforms DataSet's 100 %.

On the memory usage, SqlDataReader performs better than DataSet. The graph for total available memory on the server shows that DataSet consumes twice as much memory as SqlDataReader.

As mentioned in section 4.3.2, SqlDataReader provides a stream of data from a data source. And since the data is not built in memory, this helps reduce memory usage on server. So in accordance with both theory and our test results, the SqlDataReader method is a good choice for retrieving large amount of data from a SQL database.

9.4 Data request over network

We first tested SOAP to find out what number of rows, or amount of data, is most efficient to retrieve in one call. The results show that CPU usage in percentage increases with increasing amounts of data retrieved in one call. When 15000 rows are retrieved in one call, we see the processor time increases significantly. The differences between 10 to 1000 rows are minor.

It was expected that fetching the data would be more time consuming when fetching a small number of rows in one call. And that tendency is clear. The reason 1000 and 5000 rows use approximately the same amount of time might be that the counters are sampled only once every 5 second.

When it comes to memory use, retrieving 10 and 100 rows performs almost similarly well, with approximately 20 MB reserved memory. The memory usage increases with increased number of rows fetched in one call. But the total available memory on the server does not differ much from 10 to 1000 rows.

To find the number of rows to retrieve in one call we looked at CPU, memory and time used. Performance is best for both CPU and memory with small data amounts. But the time saved by increasing the amount of data is also important. So when considering all three variables we decided that 1000 rows was the best amount of data to transfer in on call.

We then tested both SOAP and DIME streaming with 1 to 5 clients. When using SOAP we retrieve the entire database but split the call into multiple request-response calls, were 1000 rows is retrieved in one call. The CPU usage for DIME is steady at about 60 % for 3-5 clients, while the SOAP graphs are spiky and vary between 45-75 %. For 1 client, DIME is steady at 45 %, and SOAP relatively steady at about 35 %.

On memory use, DIME performs considerably better than SOAP. DIME uses approximately 16 MB of memory, independent of the number of clients. The memory use for SOAP increases steadily from approximately 30 MB for 1client to 65 MB for 5 clients. This indicates that the use of memory will increase for a SOAP implementation if more clients are connected at the same time.

Time measurement tests show that regardless of the amount of data fetched, DIME use less time processing the request. It also shows, as we predicted, that SOAP N-Request was the most time consuming method. With increasing amounts of data fetched from server, time measured for SOAP N-Request increase more rapidly then the other methods.

As mention in section 8.4.2, the peak in CPU usage at the start of each curve in the SOAP test, is caused by the WebService being compiled and made ready to process. This disfavors SOAP as this is not the case for DIME.

There is not much difference in CPU usage for SOAP and DIME. And if we disregard the CPU usage spike at the start of the SOAP curves, time used for SOAP and DIME is almost equal. We therefore conclude that there are no significant differences in CPU and time usage between SOAP and DIME, except from the fact that DIME has steady CPU usage while SOAP is spikier.

We also conclude, based on both theory and testing, that the significant difference in server resource strain between SOAP and DIME lies in the memory usage. This becomes very apparent when multiple requests are being processed.

9.5 Feasibility discussion

9.5.1 Alternatives to streaming

As we mentioned in section 5.2.1, sending an URI to where a serialized XML document can be retrieved sidesteps the entire problem this part of the thesis focuses on. SOAP is not used for streaming, simply for setting up the connection. This solution requires the client to handle procedures deviating from standard SOAP: the client must be programmed to handle this specific scenario. In a distributed environment with open standards, this is unacceptable.

A single request multiple response message pattern seems to be the best alternative for streaming. Sending multiple responses allows the reply to be divided into several parts, and therefore limits the amount you have to build in memory. As HTTP does not support this message pattern, an alternative protocol must be agreed upon and development tools must include support for this protocol.

Multiple requests multiple responses is another feasible alternative. As in the "sending an URI" alternative, this requires the client to have software especially written for this purpose. And it is therefore not a recommended solution. In its simplest form, the users can agree to not request large amounts of data. This is perhaps the best short term solution to the problem.

9.5.2 Alternative transfer technologies

Combining SOAP with different alternative transfer technologies include different commitments and rewards.

SwA use the most common protocol for transferring SOAP messages: HTTP. It encapsulates the messages in a MIME multipart/relates messages. SwA was defined to allow SOAP to transfer non XML data (e.g. pictures and sound) and therefore does not offer any improvements or solutions to the streaming problem.

DIME offers both an alternative session layer to HTTP, and a way of transferring attachments with SOAP. As mentioned, even though Visual Studio.NET does not support it, DIME on top of TCP can provide streaming capability as well as N-Responses. Furthermore it supports chunking, which can be useful in some scenarios. See section 4.5.2.

BEEP substitutes HTTP, and can potentially enable both streaming and N-Response capability. As with SwA, it uses MIME for transferring attachments.

The main difference between DIME and MIME is that MIME use text to specify and separate message parts, while DIME use predefined headers which specifies the message. We recommend that DIME be used in the future for the following reasons: Predefined headers are much more suitable for streaming than text recognition. It allows for faster parsing with less memory use, and avoids tricky situations where the string boundaries can be used illegally as normal text, especially since SOAP intermediaries may process and change the message. This helps reduce the risk of interoperability between implementations.

9.5.3 SOAP 1.2

As mentioned in section 2.4.5, a general attachment feature has been specified for SOAP 1.2, but to no specific technology. At least by W3C. With SOAP 1.2, a possibility for streaming SOAP opens up. DIME and MIME are potential technologies for dealing with attachments.

9.6 Future work

The problem with strain on server resources when transferring large XML objects with SOAP has several feasible solutions. They do, however, require coordinated effort. The hard issues to tackle are not technological, but strategic. All, or at least a large majority, of vendors must agree upon a new standard.

Vendors must agree upon one of two possible solutions:

- Either sorting out the protocol bindings in SOAP 1.1. This can include rewriting the HTTP binding or replacing HTTP with another protocol as default for WebServices.
- Or supporting the streaming option in SOAP 1.2.

Replacing the default protocol binding for SOAP 1.1 opens for possibilities of both streaming and Request/N-Response MEP.

In this thesis, we used Microsoft's ASP.NET WebMethods. This environment does not support streaming or Request/N-Response MEP whatsoever. To solve the server resource strain for this environment, Microsoft must include support for these features in future releases.

Our belief is that the problem will be resolved with SOAP 1.2. Not resolving this problem will exclude too many possible users. This includes not only users with need to build multiple 50-100 MB XML documents on servers, but also systems with very limited resources such as Pocket PC and Personal Digital Assistant (PDA).

10 Conclusion

In SOAP, the entire XML object is generated on the server before it is returned to the client. This puts unnecessary strain on server systems in terms of both memory and CPU. This thesis deals with compression and streaming of large XML documents, including a feasibility study on why SOAP does not allow streaming, possible solutions to the problem and outlining alternative transfer methods. Furthermore, compression techniques for a streaming SOAP environment were evaluated, as well as performance of streaming versus alternative transfer methods.

Testing gave no overall compression technique winner, but the specialized XML compressors perform slightly better than the traditional text compressors. Despite this, we recommend bzip2 for use in a streaming SOAP environment. This is due to the fact that the gain in using specialized XML compressors does not outweigh text compressors 'ability to handle chunks and streams where as XML compressors can only compress intact documents.

Further testing showed that dividing a SOAP request into multiple requests eased server strain, but increased the time used to conduct the transfer. SOAP N-Request/N-Response and DIME were further tested to shed light on the difference in performance between a streaming technique and the alternative transfer method. As predicted, streaming causes less strain on server resources, in particular memory usage. This became especially apparent with multiple clients connecting.

The feasibility study concluded that SOAP itself allows streaming of responses, but the HTTP binding does not. SOAP's HTTP binding specifies the issue of a HTTP fault code in case of a SOAP processing error, meaning the processing must be completed before a HTTP code legally can be issued.

The study further concluded that there are a few possible solutions to overcoming the server resource problem. A single request with a multiple response pattern is one alternative to streaming. But as HTTP does not allow this, multiple requests multiple response is the best short term alternative. To achieve streaming or single request multiple response message pattern, SOAP must be fitted on top of another protocol than HTTP. BEEP and DIME were considered, and DIME was recommended since a binary format is better suited for streaming.

The hard issues to tackle, in order to solve the problem of server resource strain due to SOAP lack of streaming capability, are not technological but strategic. Future work includes vendors agreeing upon a standard. This can either be rewriting the HTTP binding, replacing HTTP with another protocol as default for WebServices or support the streaming option in SOAP 1.2.

11 Abbreviations

ADO:	ActiveX Data Object
BEEP:	Blocks Extensible Exchange Protocol
BWT:	Burrows-Wheeler Transform
COM:	Component Object Model
CPU:	Central processing Unit
DIME:	Direct Internet Message Encapsulation
DTD:	Document Type Definition
GUI:	Graphical User Interface
HTTP:	Hyper Text Transfer Protocol
ICT:	Information and Communication Technology
IEEE:	Institute of Electrical & Electronics Engineers
IETF:	The Internet Engineering Task Force
IP:	Internet Protocol
LZ77:	Lempel-Ziv 1977
MB:	Mega Bytes
MEP:	Message Exchange Pattern
MIME:	Multipurpose Internet Mail Extensions
MSDN:	Microsoft Developer Network
OS:	Operating System
OSI:	Open Systems Interconnection
PDA:	Personal Digital Assistant
Perfmon:	Performance Monitor
PPM:	Prediction by Partial Match
RAM:	Random Access Memory
RFC:	Request For Comment
RPC:	Remote Procedure Call
SOAP:	Simple Object Access Protocol
SQL:	Structured Query Language
SwA:	SOAP Messages with Attachments
TCP:	Transmission Control Protocol
TV:	Tele Vision
UPD:	User Datagram Protocol
URI:	Uniform Resource Identifier
UUID:	Universal Unique Identifier
WITSML:	Wellsite Information Transfer Standard Markup Language
WSE:	Web Services Enhancements
W3C:	World Wide Web Consortium

12 References

All the hyperlinks pointing to references are valid as of May 2003.

[1] The Internet Engineering Task Force
http://www.ietf.org

[2] World Wide Web Consortium
http://w3.org

[3] The W3C
Simple Object Access Protocol (SOAP) 1.1, W3C Note, May 2000
http://www.w3.org/TR/SOAP/

[4] The WITSML project
http://www.witsml.org/

[5] The W3C
Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, October 2000
http://www.w3.org/TR/REC-xml

[6] XMLPPM: XML-Conscious PPM Compression
http://www.cs.cornell.edu/people/jcheney/xmlppm/xmlppm.html

[7] The IETF
The Blocks Extensible Exchange Protocol Core, RFC3080, March 2001
http://www.ietf.org/rfc/rfc3080.txt

[8] The IETF
Mapping the BEEP Core onto TCP, RFC3081, March 2001
http://www.ietf.org/rfc/rfc3081.txt

[9] The IETF
Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP), RFC3288, June 2002
http://www.ietf.org/rfc/rfc3288.txt

[10] The W3C
SOAP Messages with Attachments, W3C Note, December 2000
http://www.w3.org/TR/SOAP-attachments

[11] MSDN
Direct Internet Message Encapsulation (DIME), Internet Draft, June 2002
Expired
http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt

[12] The IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045, November 1996
http://www.ietf.org/rfc/rfc2045.txt

[13] The IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Media Types, RCF2046, November 1996
http://www.ietf.org/rfc/rfc2046.txt

[14] The IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Message Header Extensions for Non-ASCII Text, RFC2047, November 1996
http://www.ietf.org/rfc/rfc2047.txt

[15] The IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Registration Procedures, RFC2048, November 1996
http://www.ietf.org/rfc/rfc2048.txt

[16] The IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Conformance Criteria and Examples, RFC2049, November 1996
http://www.ietf.org/rfc/rfc2049.txt

[17] The IETF
The MIME Multipart/Related Content-type, RFC2387, August 1998
http://www.ietf.org/rfc/rfc2387.txt

[18] MSDN
Using Web Services Enhancements to Send SOAP Messages with Attachments, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsedime.asp

[19] Rich Salz
Beep BEEP!
http://www.xml.com/pub/a/2002/10/16/ends.html

[20] Paul Krill
SOAP 1.2 spec takes next step
http://www.infoworld.com/article/02/12/19/021219hnsoapadvance_1.html

[21] Jeannine Hall Gailey
Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation, MSDN
http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx

[22] Andy Neilson
W3C's discussion boards
http://lists.w3.org/Archives/Public/xml-dist-app/2000Dec/0197.html

[23] The IETF
Hypertext Transfer Protocol -- HTTP/1.1, RFC2616, June 1999
http://www.ietf.org/rfc/rfc2616.txt

[24] Datacompression.info
http://datacompression.info/

[25] J. Ziv and A. Lempel
A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, Vol 23, pages 337-343, May 1977.

[26] D.A.Huffman.
A method for construction of minimum-redundancy codes, Proceeding of the IRE, Volume 40, Number 9, September 1952, pages 1098-1101

[27] M. Burrows and D.J. Wheeler.
A block-sorting lossless data compression algorithm, Technical report, Digital Equipment Corporation, Palo Alto, California, 1994.

[28] PKWARE Inc
http://www.pkware.com/

[29] Dmitry Shkarin
PPM: one step to practicality, Data Compression Conference, 2002

[30] James Snell, Doug Tidwell and Pavel Kulchenko
Programming Web Services with SOAP, Published by O'Reilly & Associates Inc. January 2002. ISBN 0-596-00095-2

[31] Mark Nelson, Jean-Loup Gailly
The Data compression book, Published by M&T Books. 1996. ISBN 1-55851-434-1

[32] Elliotte Rusty Harold
XML Bible, 2nd Edition, Published by Hungry Minds, Inc. 2001. ISBN 0-7645-4760-7

[33] The W3C
SOAP Version 1.2 Part 0: Primer, W3C Proposed Recommendation, May 2003
http://www.w3.org/TR/soap12-part0/

[34] The W3C
SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation, May 2003
http://www.w3.org/TR/soap12-part1/

[35] The W3C
SOAP Version 1.2 Part 2: Adjuncts, W3C Proposed Recommendation, May 2003
http://www.w3.org/TR/soap12-part2/

[36] The W3C
SOAP 1.2 Attachment Feature, W3C Working Draft, September 2002
http://www.w3.org/TR/soap12-af/

[37] The IETF
DEFLATE Compressed Data Format Specification version 1.3, RFC1951, May 1996
http://www.ietf.org/rfc/rfc1951.txt

[38] The bzip2 and libbzip2 official home page
http://sources.redhat.com/bzip2/

[39] PKWARE Inc
http://www.pkware.com

[40] XMill An Efficient Compressor for XML
http://www.research.att.com/sw/tools/xmill/

[41] The IETF
Transmission control protocol, RFC0793, September 1981
http://ietf.org/rfc/rfc0793.txt

[42] beepcore.org
http://beepcore.org/

[43] The SOAP Profile
http://www.clipcode.org/beep/soap/

[44] Microsoft® Visual Studio®
http://msdn.microsoft.com/vstudio/

[45] MemBoost
http://www.memboost.50g.com/

[46] Neel Sundaresan and Reshad Moussa
Algorithms and Programming Models for Efficient Representation of XML for Internet Applications
http://www10.org/cdrom/papers/542/

[47] Schema for the SOAP/1.1 envelope
http://schemas.xmlsoap.org/soap/envelope/

[48] Schema for the SOAP/1.1 encoding
http://schemas.xmlsoap.org/soap/encoding/

[49] J. G. Cleary and I. H. Witten.
Data compression using adaptive coding and partial string matching.
IEEE Trans. Comm., COM-32(4):396-402, 1984.

[50] Johan Jeuring and Paul Hagg.
Generic Programming for XML Tools. Utrecht University. Technical report UU-CS-2002-023. 2002.

[51] XComprez - A compressor for XML documents
http://www.cs.uu.nl/research/projects/generic-haskell/xmltools/XComprez

[52] Hartmut Liefke, Dan Suciu
XMILL: An Efficient Compressor for XML Data. SIGMOD Conference 2000: 153-164.

[53] Intelligent Compression Technologies.
http://www.ictcompress.com/

13 Appendices

Appendix A – XML compressors

Appendix B – Test applications

Appendix C – Test results

Appendix D – Source code. CD on the back cover.

Appendix A - XML compressors

A.1 - Millau

Introduction

The Wireless Application Protocol (WAP) defines a format to reduce the transmission size of XML documents with no loss of functionality or semantic information. In the paper "Algorithms and Programming Models for Efficient Representation of XML for Internet Applications" [46], they describe Millau who extends this format, and also improve on the compression algorithm. It separates structure compression from text compression. It also takes advantage of the schema and data type information, to achieve better compression on XML documents. To work well with the XML standards, it defines APIs equivalent to the tree model of the Document Object Model (DOM) and the event and streaming model of the Simple API for XML (SAX), to work with encoded XML documents.

Architecture

Millau starts with a wide implementation of WAP Binary Extended Markup Language (WBXML), and extends it with separation of structure and content. Then it encodes the structure part using the WBXML encoding and the content using standard text compression techniques.

The WBXML content encodes the tag names and the attributes names and values with tokens (a token is a single byte). These tokens are split into a set of overlapping "code spaces". There are two classifications of tokens, global tokens and application tokens. Global tokens are assigned a fixed set of codes in all contexts and are precise in all situations. Application tokens have a context-dependent meaning and are split into two overlapping "code spaces", the "tag code space" and the "attribute code space".

The tag code space represents specific tag names. The attribute code space is split into two numeric ranges representing attribute prefixes and attribute values. The Attribute Start token (with a value less than 128) indicates the start of an attribute and may optionally specify the beginning of the attribute value. The Attribute Value token (with a value of 128 or greater) represents a well-known string present in an attribute value. Unknown attribute values are encoded with string, entity or extension codes.

In the Millau format on the other hand, an Attribute Start token is followed by a single Attribute Value token, string, entity, or extension token. So there is no need to split the attribute token numeric range into two ranges (less than 128 and 128 or greater). Because each time the parser encounters an Attribute Start token followed by a non-reserved token, it knows that this non-reserved token is an Attribute Value token, and that it can be followed only by an end token or another attribute start token. Thus, instead of two overlapping code spaces, we have three overlapping code spaces:

- Tag code space as defined in the WAP specification.
- Attribute start code space where each page contains 256 tokens.
- Attribute value code space where each page contains 256 tokens.

In WBXML format, character data is not compressed. It is transmitted as strings inline, or as a reference in a string table which is transmitted at the beginning of the document. In Millau, character data can be transmitted on a separate stream. This allows separation of the content from the structure so that a browser can separately download the structure and the content or just a part of each. This further allows compression of the character data using traditional compression algorithms like deflate. In the structure stream, character data is indicated by a special global token (STR or STR_ZIP) which indicates to the Millau parser that it must switch

from the structure stream to the content stream if the user is interested in content and whether the content is compressed (STR) or uncompressed (STR_ZIP). Optionally, the length of the content is encoded as an integer in the structure stream right after the global token (STR_L or STR_ZIP_L). If the length is not indicated, the strings contained in the structure must terminate with an end of string character or a null character.

Conclusion

Millau is design as a binary XML compression method that is schema aware, and is very good for compressing XML documents less than 5kByte.

The biggest drawback is that it does not perform as good as traditional text-compression algorithms for large XML files.

A.2 - XComprez

Introduction

XComprez was developed by Johan Jeuring and Paul Hagg at Utrecht University. Their paper, "Generic Programming for XML Tools" [50], shows how generic programming can be used when writing XML tools, and the benefits this gives.

The program is open source, and can be downloaded at [51]. It does require some additional software to be installed, these are also open source.

Architecture

XComprez distinguishes between four different components:

- A component that translates a DTD to a data type: the data type is obtained from a DTD, together with functions for reading and writing valid XML documents to and from a value of the generated data type.

- A component that separates a value of any data type into its structure and its contents: The content is obtained by extracting all PCData and all CData from the document. The content of a value of a data type is obtained by extracting all strings from the value. And the path to the string through constructor and record label names is also recorded. The structure from an XML document is obtained by removing all PCData and CData from the document. The structure of a value of a data type is obtained by replacing all strings by units. This obtains a value of a new data type, in which occurrences of the type String have been replaced.

- A component that encodes the structure replacing constructors by bits: If "n" is the number of constructors of a data type. Then a list of $\log_2 n$ bit represents a specific constructor.

- A component for compressing the contents: At the current stage XComprez uses a zip variant for compressing the content.

A preliminary version has also been released, which analyzes some documents that are valid to the DTD, count the number of different elements and apply Huffman coding. This can further improve the compressing.

Conclusion

XComprez is an overall strong compressor. It has no specific scenario where it is especially strong.

XComprez assumes that the input document is valid according to a given DTD. Knowledge about the DTD is used to enhance and speed up the compression. But it also limits compression to files with a valid DTD. This goes for both the compressor and the decompressor.

A.3 - XMill

Introduction

XMill and XDemill are XML compression and decompression tools developed by Dan Suciu and Hartmut Liefke and described in the paper "XMill: an efficient Compressor for XML Data" [52]. The tools are not based on a new compression algorithm, but rather propose design an architecture which leverages existing compression algorithms and tools to compress XML data.

Architecture

XMill is based on Gzip in addition to a few simple data type specific compressors. The XMill compressor applies three principles to compress XML data:

- Separate structure from data: The structure consists of XML tags and attributes: it forms a tree. The data consists of a sequence of items (strings) representing element contents and attributes values. The structure and the data are compressed separately.

- Group data items with related meaning: Data items are grouped into containers, and each container is compressed separately. For example, all <name> data items form one container, while all <phone> items form a second container. This is a generalization of a well-known principle in relational databases: column- wise compression is better than row-wise compression.

- Apply different compressors to different containers: Some data items are text, others are numbers, while others may be DNA sequences. XMill applies different specialized compressors (semantic compressors) to different containers.

The XML file is parsed by a SAX parser that sends tokens to the path processor. Every XML token is assigned to a container. Tags and attributes, forming the XML structure, are sent to the structure container. Data values are sent to various data containers, according to the container expressions, and containers are compressed independently. Before entering the container, a data value may be compressed with an additional semantic compressor.

The core of XMill is the path processor that determines how to map data values to containers. The user can control this mapping by providing a series of container expressions on the command line. Containers are kept in a main memory window of fixed size (the default is 8 MB). When the window is filled, all containers are compressed with Gzip, stored on disk, and the compression resumes. In effect this splits the input file into blocks that are compressed independently.

Users can associate semantic compressors with containers. A few atomic semantic compressors are predefined in XMill, like binary encoding of integers, differential compressors, etc. The decompressor XDemill is simpler. After loading and unzipping the containers, the decompressor

parses the structure container, invokes the corresponding semantic decompressor for the data item and generates the output.

Conclusion

A limitation of XMill is that it only achieves greater compression ratios than traditional text-compression methods if dealing with data larger than approximately 20,000 bytes. But users with detailed XML knowledge can define their own compressors (for specific containers) and algorithm to improve performance, based on DTD conventions or XML-schema rules.

A second major limitation of XMill is that it is not designed to work with a query processor, hence integration of XMill's decompressor and a DB query engine.

A.4 - XMLPPM

Introduction

XMLPPM is a data compression program that compresses XML. It is a combination of the well-known Prediction by Partial Match (PPM) algorithm [49] for text compression, and an approach to modeling tree-structured data called Multiplexed Hierarchical Modeling (MHM). The XMLPPM source code is part of a project at Sourceforge on XML compression.

Architecture

XMLPPM is based on MHM and PPM modeling. The technique employs two basic ideas. Multiplexing several text compression models based on XML's syntactic structure (one model for element structure, one for attributes, and so on), and injecting hierarchical element structure symbols into the multiplexed models.

PPM models maintain statistics concerning which symbols have been seen in which contexts of preceding symbols. For each symbol, the model is used to estimate a probability range. This probability range is used to transmit the symbol using arithmetic coding. Then, the model is updated to indicate the symbol has been seen in the context. Probability estimation works as follows: If the symbol has been seen in the longest matching context, then the probability is the relative frequency in the context. Otherwise, an "escape symbol" is encoded, and the next longest context is tried, and so on. The decoder maintains the same model and uses symbols seen so far and escapes symbols to decode incoming symbols and update its model.

If neighboring symbols from different syntactic classes are drawn from distinct independent sources, the multiplexed models adapt to these distributions better than a single model can and prediction improves. However, sometimes symbols are strongly correlated with nearby symbols in different syntactic classes. In this case, multiplexing harms compression by breaking up dependences.

A common case for these dependencies is for the enclosing element tag to be strongly correlated with enclosed data. MHM exploits this by injecting the enclosing tag symbol into the element, attribute, or string model immediately before an element, attribute, or string is encoded. "Injecting" a symbol means telling the model that it has been seen but not explicitly encoding or decoding it.

Conclusion

The XMLPPM compress quite well on structured data, but is slow.

The MHM model is limited to one level of hierarchical context. More element context helps considerably in compressing structured data, but is harmful in compressing textual data and is slow. Furthermore, compressing structured data well required using the considerably slower PPM*.

A.5 - XML-XPress

Introduction

XML-XPress™ is a part of Intelligent Compression Technologies' (ICT) "XPress™ Suite of Compression Engines", which includes tools for compressing .doc files, .xml files, etc. A limited version of XML-XPress™ is available for free at the company's web site [53].

As XML-XPress is a commercial product, the technical information released about its compression techniques is limited to what is used for promotion.

Architecture

XML-XPress is a schema specific XML coder, meaning it utilizes its knowledge about a specific XML document to improve the compression of that document. A DTD or schema can define the structure of an XML document. When the schema is known, XML tags can be encoded very efficiency.

For example, if an element contains two sub-elements (A and B), the decoder can reconstruct the tags without needing any information from the encoded file: it knows exactly where the start and end tags of both elements A and B will be located.

Also, only a binary decision needs to be included in the encoded file to determine which of A or B is present.

XML-XPress can further take advantage of sample documents. The encoding of a binary decision such as (A and B) can be done more efficiently when the compression knows the average frequency at which the two options occur in a specific context.

Schemas also provide information about the data types of the different element data. This information allows specific compression routines for the different element data.

XML-XPress use Schema Model Files (SMF). Information from a schema or DTD together with sample XML files are used to generate the SMF. This is done by ICT.

Conclusion

XML-XPress is an overall strong compressor. It has no specific scenario where it is especially strong.

It is schema specific and this makes it a very effective and fast compressor, but it has its disadvantages. Prior to communication, both the compressor and decompressor need the SMF for every type of XML file.

In the absence of such a SMF, XML-XPress resorts to using a general-purpose encoder, and the outstanding compression performance is lost.

Appendix B - Test applications

This appendix describes the applications used for testing. The source code can be found in Appendix D on a CD at the back cover.

B.1 - Compression tester

Compression tester is used to manage the compressors. The compressors were all command line tools, so making an application to manage them saved us from a lot of unnecessary work. The time measurement would be really difficult without this application.

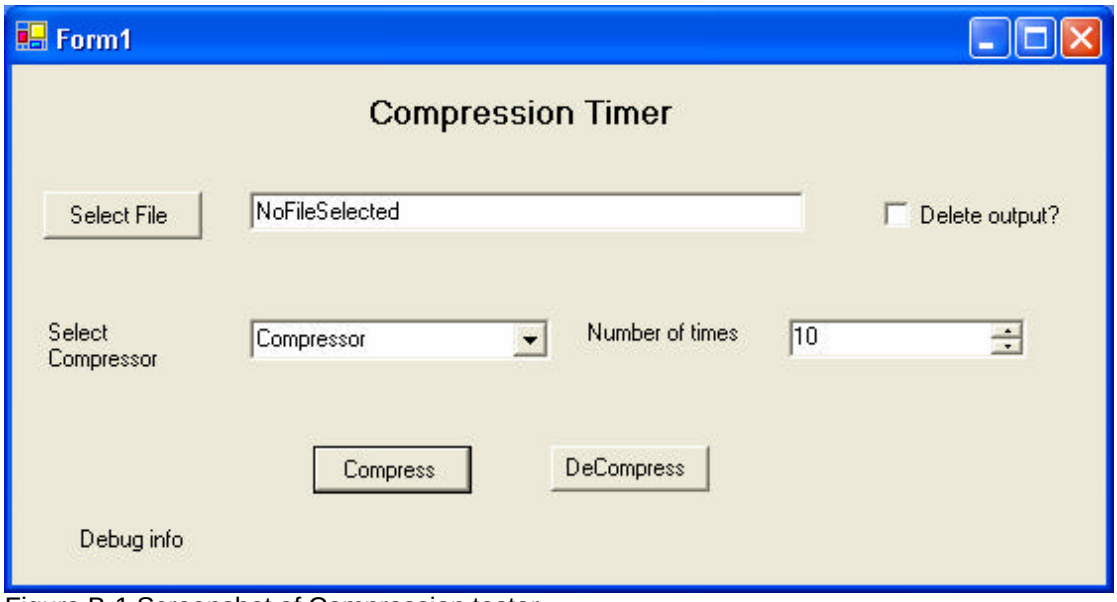


Figure B-1 Screenshot of Compression tester

B.2 - Generator

Generator is used to fill the test database "XMLLog" with approximately 7500 rows of data each time it is executed. Values for the attribute "TIMEDATE" is generated by setting in the date and hour for every day in 2003. "LOGDATA" values are generated randomly.

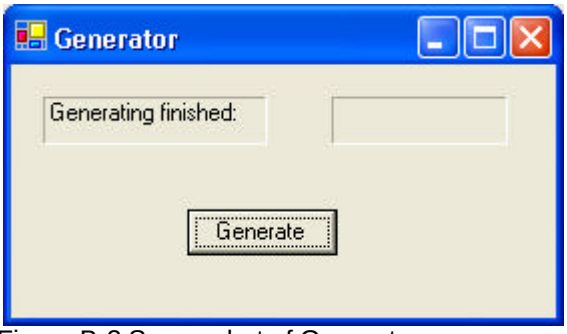


Figure B-2 Screenshot of Generator

B.3 - Database stream tester

Database stream tester is used to test streaming from a database. It is possible to choose two different methods for fetching the entire database. DataSet who builds the data in memory before it is written to a file, and SqlDataReader who generates a stream of rows that is written to a file.

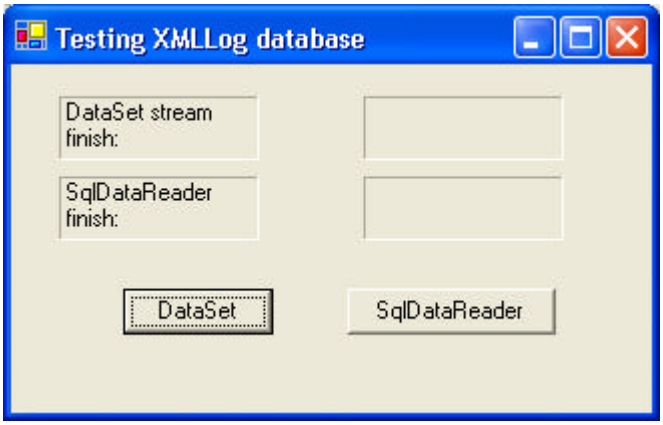


Figure B-3 Screenshot of Database stream tester

B.4 - SOAP N-Request tester

SOAP N-Request tester is used to test SOAP over a network. The client connects to a WebService on the server and requires a number of rows. The WebService fetches data from the database and puts it in a dataset, which is sent back to the client.

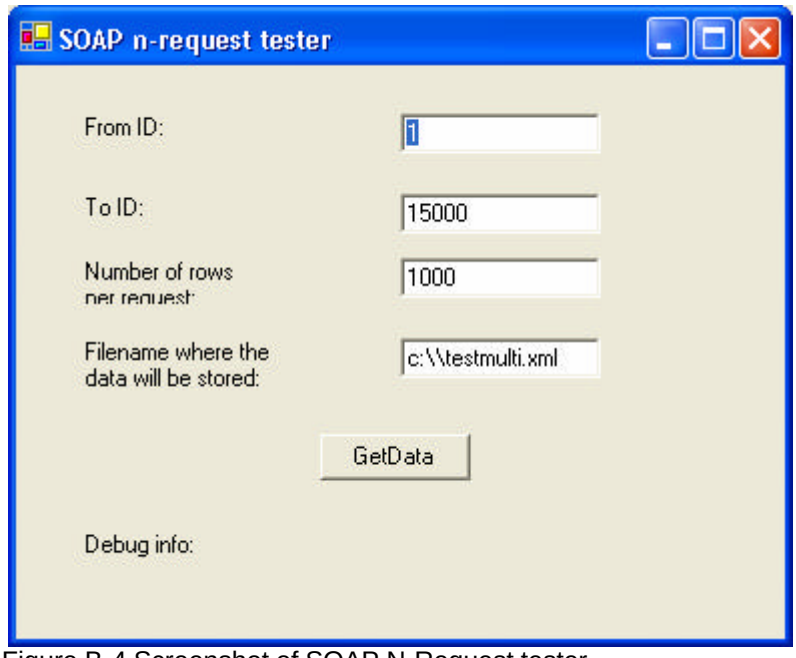


Figure B-4 Screenshot of SOAP N-Request tester

B.5 - DIME tester

DIME tester is used to test DIME over a network. The client connects to a TCP listener on server and requests a number of rows from the database. The server fetches them in a SqlDataReader stream, which is streamed back to the client in a chunked DIME message.

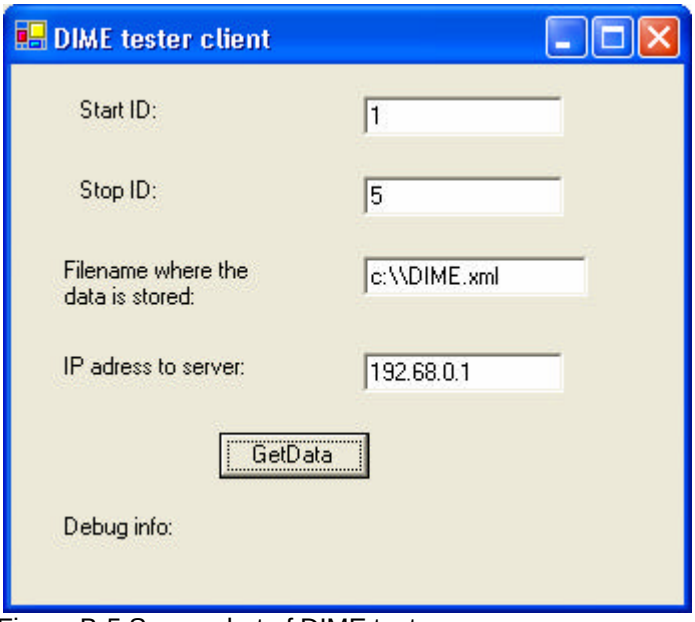


Figure B-5 Screenshot of DIME tester

Appendix C - Test results

C.1 - Compression test

C.1.1 Compression results

Compression size in bytes

XML document	original size	bzip2	PKZip	XMill	XMLPPM
capClient	405	268	378	293	200
wellbore_no_xsl	2642	734	793	791	607
mudLog_no_xsl	7392	1923	2094	2454	1765
rig_no_xsl	14782	3216	3579	4250	2905
SOAPtest0	34738	11370	9597	9515	10860
SOAPtest1	101838	31034	25377	25196	31202
SOAPtest2	505743	148272	121431	119969	175337
SOAPtest3	1004172	299437	251545	248357	375264
SOAPtest4	2001975	630952	553118	546403	774608
SOAPtest5	2995965	987183	907145	895423	1167435

Compression size in %

XML document	bzip2 %	PKZip %	XMill %	XMLPPM %
capClient	66,2	93,3	72,3	49,4
wellbore_no_xsl	27,8	30,0	29,9	23,0
mudLog_no_xsl	26,0	28,3	33,2	23,9
rig_no_xsl	21,8	24,2	28,8	19,7
opsReport_no_xsl	32,7	27,6	27,4	31,3
SOAPtest1	30,5	24,9	24,7	30,6
SOAPtest2	29,3	24,0	23,7	34,7
SOAPtest3	29,8	25,0	24,7	37,4
SOAPtest4	31,5	27,6	27,3	38,7
SOAPtest5	33,0	30,3	29,9	39,0

Compression time in seconds

	original size	bzip2	PKZip	XMill	XMLPPM
capClient	405	0,06	0,67	0,06	0,07
wellbore_no_xsl	2642	0,04	0,64	0,05	0,06
mudLog_no_xsl	7392	0,04	0,77	0,05	0,11
rig_no_xsl	14782	0,10	0,67	0,06	0,16
SOAPtest0	34738	0,12	0,67	0,12	0,41
SOAPtest1	101838	0,33	0,70	0,11	0,90
SOAPtest2	505743	1,40	0,84	0,41	4,83
SOAPtest3	1004172	2,74	1,06	0,72	10,26
SOAPtest4	2001975	5,04	1,46	1,28	22,34
SOAPtest5	2995965	7,22	1,98	2,28	35,54

C.1.2 Decompression results

Decompression size in bytes

	original size	bzip2	PKZip	XMill	XMLPPM
capClient	405	405	405	405	405
wellbore_no_xsl	2642	2642	2642	2642	2642
mudLog_no_xsl	7392	7392	7392	7392	7392
rig_no_xsl	14782	14782	14782	14782	14801
SOAPtest0	34738	34738	34738	34738	34969
SOAPtest1	101838	101838	101838	101838	102514
SOAPtest2	505743	505743	505743	505743	509094
SOAPtest3	1004172	1004172	1004172	1004172	1010823
SOAPtest4	2001975	2001975	2001975	2001975	2015226
SOAPtest5	2995965	2995965	2995965	2995965	2995965

Decompression size in %

	bzip2 %	PKZip %	XMill %	XMLPPM %
capClient	100,0	100,0	100,0	100,0
wellbore_no_xsl	100,0	100,0	100,0	100,0
mudLog_no_xsl	100,0	100,0	100,0	100,0
rig_no_xsl	100,0	100,0	100,0	100,1
opsReport_no_xsl	100,0	100,0	100,0	100,7
SOAPtest1	100,0	100,0	100,0	100,7
SOAPtest2	100,0	100,0	100,0	100,7
SOAPtest3	100,0	100,0	100,0	100,7
SOAPtest4	100,0	100,0	100,0	100,7
SOAPtest5	100,0	100,0	100,0	100,0

Decompression time in seconds

	original size	bzip2	PKZip	XMill	XMLPPM
capClient	405	0,07	0,60	0,07	0,05
wellbore_no_xsl	2642	0,04	0,64	0,05	0,05
mudLog_no_xsl	7392	0,06	0,61	0,08	0,11
rig_no_xsl	14782	0,06	0,66	0,06	0,13
SOAPtest0	34738	0,08	0,66	0,07	0,32
SOAPtest1	101838	0,13	0,62	0,08	0,94
SOAPtest2	505743	0,50	0,64	0,10	4,75
SOAPtest3	1004172	0,73	0,78	0,17	9,99
SOAPtest4	2001975	1,35	0,84	0,29	21,82
SOAPtest5	2995965	1,96	1,00	0,49	35,10

C.2 - Database streaming results

Process CPU usage

Time	DataSet	SqlDataReader
0	0,00	0,00
2	0,00	0,00
4	0,00	0,00
6	0,00	1,00
8	2,00	4,50
10	4,50	4,02
12	2,51	22,11
14	7,54	36,68
16	16,50	7,50
18	37,50	21,00
20	9,00	29,50
22	15,00	24,50
24	47,00	39,50
26	30,15	32,66
28	71,86	18,09
30	99,50	0,50
32	32,00	0,50
34	0,00	0,00
36	0,00	0,00
38	0,00	

Total CPU usage

Time	DataSet	SqlDataReader
0	99,99998	99,99998
2	2,5	2,512563
4	1	0
6	1,5	1,5
8	4	7
10	10	8,040201
12	4,020101	33,16583
14	17,58794	49,24623
16	28	23
18	46	24
20	20	58
22	18	44,5
24	80	60
26	57,78894	54,77387
28	84,92462	37,68844
30	100	2,5
32	34	6
34	2,5	2
36	4,5	0
38	1	

Total Memory usage

Time	DataSet	SqlDataReader
0	178757632	178876416
2	178667520	178839552
4	178683904	178835456
6	178659328	178855936
8	178597888	177491968
10	176164864	175693824
12	174985216	173854720
14	173379584	169885696
16	171589632	167968768
18	169197568	166637568
20	167968768	161030144
22	166764544	158076928
24	148070400	152502272
26	139214848	148705280
28	128475136	146968576
30	121131008	148267008
32	121143296	149159936
34	122966016	149975040
36	124370944	150499328
38	125452288	

C.3 - Data request over network test

C.3.1 SOAP N-Request results

CPU usage for the web service "SOAP N-Request tester"

Time	15000 rows	5000 rows	1000 rows	100 rows	10 rows
0	0	0	0	0	0
5	0	0	0	0	0
10	0	0	0	0	0
15	8,032129	2	2,204409	11,82365	1,2
20	56,91383	38,07615	71,6	78,2	24,4489
25	28,05611	55,4	31,66333	17,03407	68,2
30	18,63727	16,43287	7,6	10,62124	16,43287
35	5,8	5,01002	32,66533	25,4509	13,42685
40	19,23848	32,4	37,6	37,87575	20,64128
45	65,53106	36,07214	30,2	36,07214	30,46092
50	96,39279	42,68537	30,26052	36,87375	32,06413
55	6,412826	17,83567	34	38,07615	37,4
60	2,4	57,11423	22	41,2	34,66934
65	0,200401	0	0	32,86573	31,2
70	1,4	0		0	32,2
75				0	30,86172
80					33
85					34,8
90					25,4509

The amount of memory reserved for the web service "SOAP N -Request tester"

Time	15000 rows	5000 rows	1000 rows	100 rows	10 rows
0	0	0	0	0	0
5	0	0	0	0	0
10	0	0	0	0	0
15	8331264	57344	1351680	10534912	57344
20	16433152	15200256	18452480	17551360	14151680
25	17883136	17809408	18849792	19877888	18944000
30	20475904	20365312	20258816	21082112	21397504
35	24117248	20766720	31133696	22364160	20996096
40	36306944	44015616	31797248	22364160	22347776
45	79355904	64348160	31903744	22364160	22347776
50	1,31E+08	71507968	31698944	22757376	21635072
55	1,3E+08	65564672	32280576	22364160	21843968
60	1,22E+08	64552960	32747520	22364160	20701184
65	1,22E+08	64552960	32747520	22364160	21524480
70	1,22E+08	64552960		22364160	21716992
75				22364160	21594112
80					21737472
85					21848064
90					21995520
95					21995520
100					21995520

Total free memory on the server

Time	15000 rows	5000 rows	1000 rows	100 rows	10 rows
0	182636544	193212416	192507904	191270912	187707392
5	183545856	192409600	191447040	189878272	187707392
10	181653504	190541824	189919232	188407808	185610240
15	176734208	184135680	189300736	181293056	185524224
20	168583168	178515968	174673920	174018560	174874624
25	165793792	174829568	174194688	171663360	169218048
30	164044800	173051904	170979328	167747584	167272448
35	155869184	170274816	156508160	164319232	165359616
40	135204864	142209024	153096192	161923072	160509952
45	89268224	118845440	150290432	159444992	156889088
50	34295808	107134976	147918848	156676096	153919488
55	27320320	111677440	145510400	154685440	149897216
60	35332096	108806144	143630336	153309184	147136512
65	35024896	108847104	143650816	151261184	142848000
70	35590144	109379584		151961600	139915264
75				152006656	136220672
80					132272128
85					128241664
90					125353984

C.3.2 SOAP N-Request 1-5 clients results

CPU usage for the web service "SOAP n-request tester"

Time	1 client	2 clients	3 clients	4 clients	5 clients
5					
10					
15	1,40562249	1,40280561	1,60320641	1,80360721	1,00200401
20	13,6	21,8436874	37,4	54,9098196	15,0300601
25	76,753507	72,9458918	57,5150301	45,0901804	73,4
30	15,4308617	17,6	20,240481	16,4	24,4488978
35	8,61723447	38,2765531	44,6	55,9118236	18,0360721
40	12,8256513	48,0961924	29,258517	34,8	56,3126253
45	20,0400802	55,511022	34,0681363	72,3446894	10,2
50	34,0681363	62,1242485	36,4729459	60,5210421	75,3507014
55	32,8657315	67,9358717	73,747495	67,5350701	53
60	24,6492986	65,9318637	68,9378758	60,3206413	71,743487
65	35,6	31,2625251	56,9138277	66,9338677	67,1342685
70	23	0	41,6833667	66,9338677	49,6993988
75	0		62,9258517	58,9178357	72,3446894
80			64,1282565	51,503006	48,0961924
85			38,8777555	66,1322645	74,5490982
90			0	59,3186373	57,7154309
95				18,4368737	70,2
100				0	49,2985972
105				0	60,1202405
110					52,2
115					32,6653307

The amount of memory reserved for the web service "SOAP n-request tester"

Time	1 client	2 clients	3 clients	4 clients	5 clients
0					
5					
10					
15	57344	57344	57344	372736	57344
20	11427840	14012416	15249408	16658432	12095488
25	17649664	17776640	18460672	18677760	18055168
30	19439616	20348928	21233664	21098496	20148224
35	20779008	34504704	42287104	47878144	29130752
40	25878528	43556864	43855872	56832000	54665216
45	31531008	43012096	47644672	55296000	62136320
50	30380032	41463808	47845376	56598528	61792256
55	32792576	41168896	47157248	54398976	64729088
60	32280576	45297664	45764608	53895168	65519616
65	30416896	41127936	43405312	55296000	60805120
70	30425088	41127936	45465600	52998144	65904640
75	30425088		47681536	53059584	62730240
80			45473792	56795136	61571072
85			43413504	57802752	61132800
90			43413504	56188928	63250432
95				56602624	60825600
100				56602624	60772352
105				56602624	60772352
110					64659456
115					62398464

C.3.3 DIME 1-5 clients results

CPU usage for "DIME tester" application.

Time	1 client	2 clients	3 clients	4 clients	5 clients
0	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00
10	0,00	0,00	0,00	0,00	0,00
15	10,60	11,22	8,02	6,61	11,02
20	24,85	22,04	22,80	11,60	30,20
25	42,60	39,00	47,29	36,87	50,00
30	45,29	49,60	50,90	51,00	61,92
35	42,89	52,30	51,10	50,20	53,91
40	0,00	51,10	57,31	60,12	57,60
45	0,00	50,90	50,90	57,20	56,51
50		21,24	63,80	58,20	55,31
55		0,00	54,11	57,72	59,52
60			40,28	58,32	56,51
65			0,00	59,20	59,92
70				56,31	59,52
75				41,60	61,32
80				0,00	59,32
85					16,83

The amount of memory reserved for the "DIME tester" application.

Time	1 client	2 clients	3 clients	4 clients	5 clients
0	9732096	9539584	9744384	9719808	9703424
5	9732096	9539584	9744384	9719808	9703424
10	9732096	9539584	9744384	9719808	9703424
15	13127680	12988416	12091392	10842112	13185024
20	15945728	15724544	15839232	13860864	16097280
25	15945728	15740928	15855616	16080896	16105472
30	15945728	15740928	15855616	16080896	16109568
35	15986688	15740928	15855616	16080896	16109568
40	16003072	15740928	15855616	16080896	16109568
45	16003072	15745024	15855616	16080896	16109568
50		15855616	15855616	16080896	16109568
55		15855616	15855616	16080896	15474688
60			15859712	16080896	15478784
65			15859712	16080896	15478784
70				16080896	15482880
75				16068608	15482880
80				16068608	15491072
85					15462400
90					15462400

C.3.4 Time measured for DIME, SOAP and SOAP N-Request

Time on server for different number of rows requested.

Rows requested	DIME	SOAP	SOAP N-Request
1	8	17	18
1000	10	19	20
5000	14	23	28
10000	18	26	37
15000	22	32	46