Abstract

This report describes a prototype middleware system for optimising transfer and processing times of XML based data between mobile, heterogeneous clients, supporting servers and context providers. The system will achieve these objectives by compressing or compacting the XML data in different ways, and using different parsing techniques.

Two such techniques are examined more thoroughly, namely tag redundancy reduction, and binary compression. These optimisation techniques are implemented in a fully functioning XML data optimising system, and their effectiveness is tested and compared.

A long term goal is discussed and considered in relation to these techniques: To develop a set of heuristic rules that will allow the system to determine dynamically which optimisation methods are most efficient at any given time based on available context data.

The prototype system described is developed in Java, with a client for mobile devices written in Java2ME.

Preface

This paper is the result of my master degree work at the Department of Computer and Information Science at the Norwegian University of Science and Technology in Trondheim during spring 2005.

I would like to thank my supervisors, Carl-Fredrik Sørensen and Alf Inge Wang for all their help, guidance and patience during this work.

Trondheim, June 2005

Anders Kristian Harang Walla

Table of contents

List of figures

List of tables

Part I: Introduction

Chapter 1: Introduction

This project aims to provide a solution for better support for XML data transfer in a mobile, heterogeneous environment. This chapter begins with the motivation for developing such a solution and goes on to describe MObile Work Across Heterogenous Systems (MOWAHS) – the larger project in which this project is encompassed. Lastly, the original problem description is presented.

1.1 Motivation

With the recent and ongoing developments in handheld computers, digital assistants, mobile phones and even computers integrated in clothing, *ubiquitous computing* [19] is increasingly and rapidly making its way into our everyday life. Wireless networks of different kinds surround us and connect people to their information sources wherever they go through a wide variety of heterogeneous devices. In this new and rapidly growing environment, we have more and more companies working as *virtual organizations*, with people distributed across several locations and time zones. In the near future, people and machines may be working safely together with the help of location-aware devices interacting with each other over wireless networks. In both these scenarios, a number of heterogeneous devices need to communicate over unreliable, wireless networks.

The heterogeneity both in devices and networking protocols that is part of the nature of ubiquitous computing, will surely lead to the need for a common format for data storage and exchange. XML is already a standard growing rapidly in popularity and usage because of its simplicity and interoperability across platforms. We can already observe the interest in support for data exchange via XML by the two major tools in software development for limited computer devices, Microsoft .NET and Java2 Micro Edition.

However, as appealing as it might be to utilize XML in this context, there are two obvious problems with doing so. Firstly, XML stores its data explicitly, with little regard to conserving space. As keeping the load on our available wireless networks as low as possible may be of significance, we do want to keep the transmitted data to a minimum. Secondly, parsing XML data is a resource-demanding operation in terms of both processing power and memory usage, both of which are often scarce on limited mobile devices.

In this project a middleware system for reducing the size of any transferred XML data and minimizing parsing time will be designed, implemented and tested in Java and Java2ME. Different techniques will be applied to achieve this, and assessed individually. Finally, the possibility of dynamically choosing which techniques to apply in a given situation based on context data will be investigated.

1.2 Project context: MOWAHS

This report is written as a part of the MOWAHS[25] research project, conducted at the Norwegian University of Science and Technology. The project is funded with NOK 5 million over four years by the Norwegian Research Council through its IKT-2010 program and it is carried out jointly by IDI's (Department of Computer and Information Science) groups for software engineering and database technology.

The project has two parts: 1) Process support for mobile users using heterogeneous devices (PC, PDA, mobile phones); and 2) support for cooperating transactions/workspaces holding work documents. This report, dealing with communication between heterogeneous devices, falls under the former.

The goals of MOWAHS are threefold:

- **G1** Helping to understand and to continuously assess and improve work processes in virtual organizations
- **G2** Providing a flexible, common work environment to execute and share real work processes and their artefacts, applicable on a variety of electronic devices (from big servers to small PDAs).
- **G3** Disseminating the results to colleagues, students, companies, and the community at large.

This project will mainly deal with the second goal of MOWAHS as it aims to make improvements to the communication between electronic devices.

1.3 Problem description

The initial task description for this thesis was as follows:

MOWAHS – Optimised support for XML in mobile environments.

XML parsing is a relatively resource-heavy operation. Your task will be to develop models, and one or more systems that use XML to communicate between mobile devices such as PDAs and mobile phones. How can we build a system based on mobile clients and proxy servers that will optimise processing time and the transfer time of XML-based information between these devices? This will continue work done in depth studies during autumn 2003/2004. The results from 2003 show resource usage (memory usage, processing times, transfer times, total times) acquired from different solutions. Is it possible to construct a set of heuristic rules for whether or not XML data should be transformed before being transferred to the mobile device based on these results? This project will build a prototype system that performs transformations on the XML files and transfers these to a J2ME/.NET-based device (PDA, mobile phone etc.)

Chapter 2: Research Agenda

This chapter presents the basis of the report - the research questions which this project aims to answer. A few hypotheses are also presented, whose validity will be challenged later through practical experiments. Finally, the research methodology that the work is based upon is described.

2.4 Research questions

From the problem description, the main question that this report will try to answer is deduced:

How can a system be built that optimises the transfer and processing of XML-based information between mobile, heterogeneous clients and supporting servers by introducing an intervening proxy server that will perform certain transformations of the XML data sent according to sensory data? Which technologies are candidates for such an optimisation, and how well do they really perform?

The following sub-questions will help lead the way to a solution to this question.

- *Current state* – Are there any current efforts to solve the question at hand? What technologies are available for a possible solution?
- *Requirements* – Which limitations and challenges exist in the context of XML data transferring and parsing in a mobile environment?
- *Solution* – How can the requirements be met? Is it possible to build a prototype system to use as a testing platform for evaluating the different techniques?
- *Evaluation* – How well do the solutions provided deal with the original problem? How can the validity of the solution be properly tested?

2.5 Research hypotheses

To further illuminate the problem at hand, a set of research hypotheses has been devised from the problem description in 1.3 and the research questions in 2.4. They give a concrete representation of the issues this report will look into.

H1 There exists a way of optimising the process of transferring (T) and parsing (P) XML data between a content provider and mobile units via an intervening proxy server with an overhead (O) in processing time small enough for the following to be true.

$$time(T_{optimised} + P_{optimised} + O) < time(T_{unoptimised} + P_{unoptimised})$$

H2 It is possible to select the optimal compression and parsing techniques dynamically in any situation according to a heuristic rule-set.

2.6 Research method

Choosing an appropriate, systematic research method is important for the successful completion of a project. In general, a *method* represents the means, procedure or technique used to carry out a process in a logical, orderly or systematic way. According to Berndtsson, Hannsson, Olsson and Lundell [8], a method, in the context of a research project, refers to an organized approach to problem-solving that includes the following five points.

i. **Collecting data** – The pre-study is an important part of the report, as it surveys the current state in the field of investigation. Available technologies for a solution are uncovered and explored thoroughly.

ii. **Formulating a hypothesis or proposition** – The research hypotheses in chapter 2.5 clearly states what the work in the report aims to cover.

iii. **Testing the hypothesis** – The knowledge gathered in the pre-study is put to use and a prototype system is built to test the hypotheses. This is set out in the part of the report under "Own Contribution".

iv. **Interpreting results** – In this section of the report the results of the testing of the hypotheses are discussed.

v. **Stating conclusions that can later be interpreted by others** – Finally, the knowledge gained from the process is extracted, and presented in the final chapter.

This method is also known as the *scientific method* and the work in this report is based upon it

Part II: State-of-the-art

Chapter 3: Challenges in mobile computing

The science of mobile computing is a relatively new one, and working in such an environment we face different challenges than the ones we are used to in a stationary setting. In the following, we will briefly look at some of the challenges [11] within mobile computing:

- **Disconnections**: Wireless networking may expose users to frequent disconnections, as opposed to traditional computer systems. This is important to bear in mind when designing software for mobile units, as these applications have to tolerate such unreliable network conditions.

- **Low bandwidth:** Even though third generation mobile networks are slowly seeing their way into the market; bandwidth is still generally very limited on mobile networks. In many cases the user is charged per volume used, so one would want to limit the bandwidth usage anyways.

- **Bandwidth variability:** The network conditions in a mobile environment can change very rapidly, and applications will have to deal with this in an appropriate manner.

- **Heterogeneous networks:** Mobile units may need to adapt to a number of various network types on the go, and even variations within a single network type.

- **Security risks:** The security of mobile units can much more easily be compromised than wired units, due to the availability and connectivity of wireless links.

- **Address migration:** Today's networks are not designed for dynamically changing addresses, and therefore mobile users on these networks must overcome the challenge of somehow establishing some kind of mobile IP.

- **Location-awareness:** There are advantages of combining mobility and computing, and location-specific information should be utilised fully by future mobile applications.

- **Low power:** Mobile devices often have limited battery capacity, so conserving power is of the essence.

- **Risks to data:** Portable units are easily lost, damaged or stolen, so steps should be taken to avoid unauthorized disclosure of information.

- **Small user interface:** Making good user interfaces on small, portable devices is always challenging.

- **Small storage capacity:** Small units often does not have a lot of available storage space, a solution to this could be to store data off the device.

Chapter 4: Java 2 Platform, Micro Edition

Java 2 Platform, Micro Edition (J2ME) was introduced by Sun Microsystems in June 1999. The platform is targeted at consumer electronics (e.g. satellite TV receiver), mobile devices (PDAs, mobile phones, pagers) and embedded devices (ATMs, driving computers etc). As shown in Figure 1, J2ME is one of the four software development platforms provided by Sun, the others being Java2 Enterprise Edition (J2EE) for large, complex systems, Java2 Standard Edition (J2SE) for normal applications and Java Card for smart cards. [5]

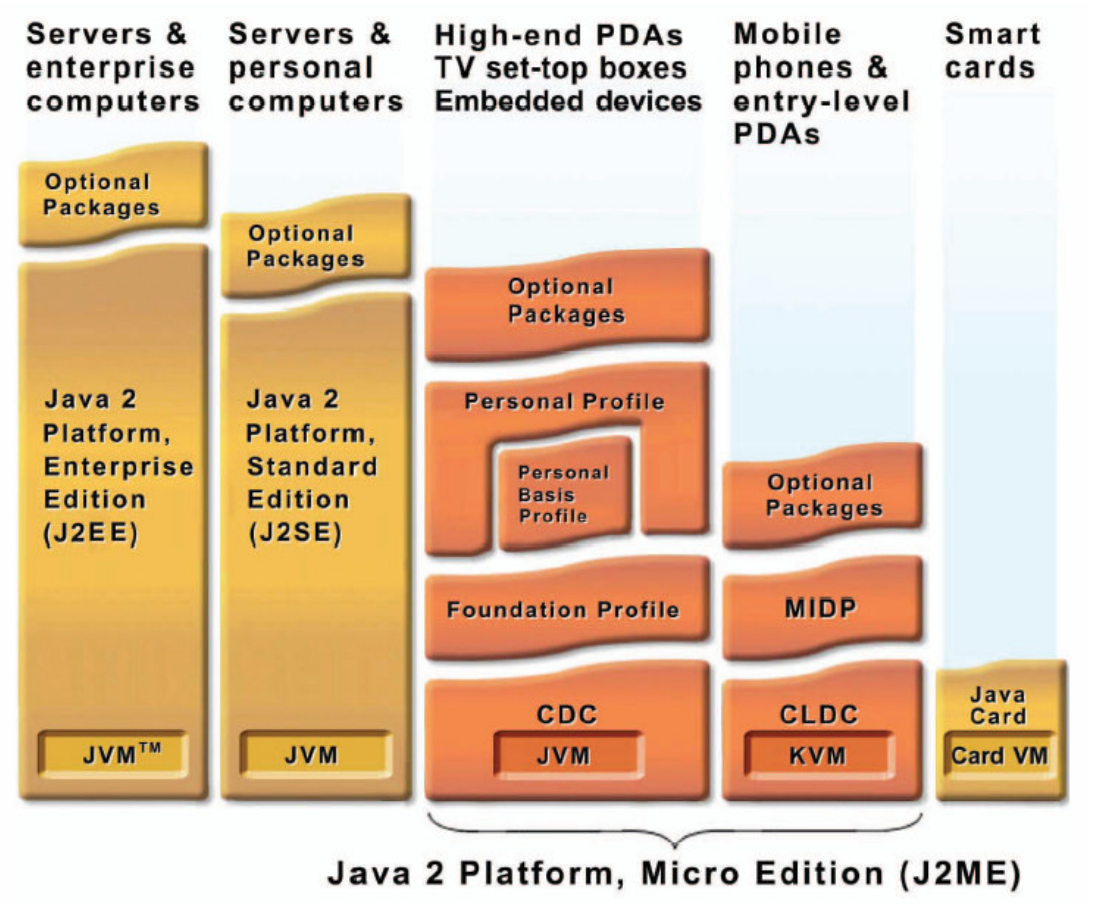

Figure 1 – J2ME and the other Java platforms

4.7 Architecture

J2ME consists of three modular and scalable layers: Java Virtual Machine, Configurations and Profiles. [6][7]

4.7.1 Java Virtual Machine

The Java Virtual Machine's (JVM) layer's main function is to communicate with the device's operating system, so every operating system will have its own custom tailored JVM. For larger, resource heavy units such as consumer electronics, this will be a standard JVM, but for smaller units like PDAs and mobile phones a lighter version has

been developed: the Kilobyte Virtual Machine (KVM). These units often have limited resources, such as 160 to 512 KB of memory, into which the standard JVM would never fit. In addition, some of the features of the JVM would not necessarily be useful to these smaller units and these features have therefore been sacrificed to minimize the size of the KVM, for example, floating-point type and AWT[1] . Instead, the LCDUI (Liquid Crystal Display User Interface), a reduced set of UI functions aimed for small devices, was introduced as a substitute for AWT. As a result, the KVM is in the range of 40 to 80 Kbytes (hence the "K" in KVM).

4.7.2 Configurations

The configuration layer lies above the JVM. This layer defines a minimum set of JVM features and core Java class libraries available on a particular category of devices. As of today, two configurations exist: *Connected Device Configuration* (CDC) and *Connected Limited Device Configuration* (CLDC). The former is designed for larger devices that are always connected, but still considered relatively resource poor compared to standard computers, such as a satellite TV receiver. CDC runs on a traditional JVM. The latter is designed for smaller, weaker units such as PDAs and mobile phones, and runs on the aforementioned KVM.

4.7.3 Profiles

The final part of the J2ME architecture is the profile. The idea of the profile was to provide an interface towards the programmer, built on top of the configuration for a certain group of units. The purpose of the profile is twofold: *Device specialization* (APIs that capture or exploit particularities of the device interface and capability) and *device portability* (APIs that behave consistently on any device supporting the profile).

The Mobile Information Device Profile (MIDP) was the first available profile. Later there were attempts at creating additional profiles, such as a profile for a certain group of PDAs, however MIDP has emerged as the de facto standard for such small devices.

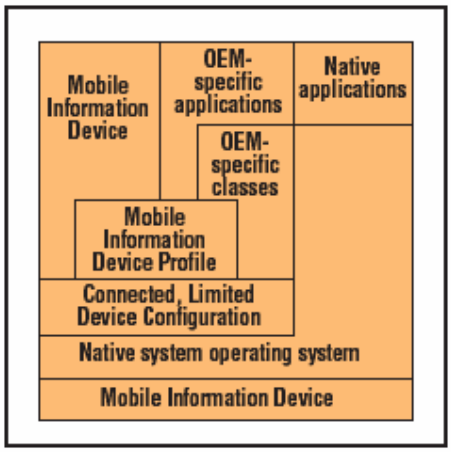

Figure 2 – J2ME architecture

MIDP provides API classes related to interface, persistence storing and network connections. It also provides a set of requirements for device manufacturers to use as

[1] Java Abstract Window Tookit

guidelines if they want their devices to be CLDC and MIDP branded, for instance the minimum display requirements are a 96 x 54 pixel screen size, and the device needs to have at least 128 Kbytes of RAM.

Figure 2 shows the architectural view of the relationship between the KVM (Native system operating system), CLDC, and MIDP layers. The immediately adjacent neighbours to the "Mobile Information Device" block show functionality what will be available to a potential mobile software developer for this device. We can see that some of the device's functions may be unavailable to the developer; a device could for example have Bluetooth support and pre-installed software that utilizes it while still not providing it to the J2ME developer. This could lead to problems when developing software for different devices because the full capabilities of the device may not be available to the developer.

MIDP 1.0 was the first profile to be available when J2ME was introduced. MIDP 2.0 was introduced in November 2002, but even today a vast amount of devices only support MIDP 1.0. This is a potential problem area for developers who want their software to be available to as many as possible but require functionality in MIDP 2.0, but should be insignificant in terms of this project since our primary scope is to use the system for testing performance on selected devices.

Chapter 5: The Extensible Markup Language (XML)

The Extensible Markup Language is a general-purpose markup language for creating special-purpose markup languages [1]. It was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996 [2]. Much like HTML, XML is a subset of the more complex SGML (Standard Generalized Markup Language). XML however, has no predefined elements known from HTML such as
 and <table>, and has a slightly stricter syntax, as its usage falls more towards structuring, describing and interchanging data. However, comparing XML to SGML, it is being said that XML offers about 90% of the functionality of SGML at some 10% of its complexity.

The design goals for XML, as laid out in the XML Specification [2] are:

1. *XML shall be straightforwardly usable over the Internet.*
   With an XML-capable browser, XML source documents must be viewable as quickly and easily as HTML documents are today.
2. *XML shall support a wide variety of applications.*
   XML usage must not be limited to only web applications, but also be available for more general use. Examples are authoring, data presentation, content analysis and validation, and database applications.
3. *XML shall be compatible with SGML.*
   This point is really a given, since XML originally was designed as a subset of SGML.
4. *It shall be easy to write programs that process XML documents.*
   XML, being a light version of SGML, has eliminated the parts of SGML that are difficult to implement, and writing an XML parser is therefore a quite trivial task. As a result of this, many different XML parsers are available today.
5. *The number of optional features in XML is to be kept to the absolute minimum, ideally zero.*
   No optional features are allowed in XML. This keeps confusion to a minimum when writing an XML parser and it also ensures that every XML parser in the world can interpret any given valid XML document.
6. *XML documents should be human-legible and reasonably clear.*
   We know from experience that being able to read the contents of a document without having to resort to a designated program to do so is a huge advantage. Making small changes to an XML document with your favourite text editor in no time is a trivial task.
7. *The XML design should be prepared quickly.*
   It was necessary to come up with a solution to HTML's problems very quickly, and the XML Working Group did so in minimal time.
8. *The design of XML shall be formal and concise.*
   This point suggests that the XML language should be easy to describe formally, thus easing parsing, and allowing the average document programmer to master the language easily.
9. *XML documents shall be easy to create.*
   Generating correct XML should be a trivial task to intelligent editors.

10. *Terseness in XML markup is of minimal importance.*
    No minimization of the markup is allowed, again to obtain readability.
    Conciseness comes second to clarity.

5.8 The XML Components

The basic concept of XML is that every *document* is composed of a series of *entities*. Each entity contains one or more *elements*, and elements can in turn be characterized by *attributes*. In the following example everything between "<staff>" and "</staff>" belongs to the staff element, and other elements (employee, name and address) are nested within this element. "id" is an attribute assigned to the employee-elements, the value "1" is assigned to the first employee, while the value "2" is assigned to the second.

```
<?xml version="1.0" encoding="UTF-8"?>

<staff>
     <employee id="1">
          <name>Anders</name>
          <address>165 Grattan Street</address>
     </employee>
     <employee id="2">
          <name>Mel</name>
          <address>42 Flowerdale Road</address>
     </employee>
</staff>
```

Figure 3 – Sample XML file

The relationship between elements and their possible attributes can be defined by a Document Type Description (DTD), and much of the beauty of XML lies in that by defining a simple DTD the user can define his own language for whichever kind of document he has to deal with. By choosing user-friendly and human-legible markup tags, the use of the tags will be easier to comprehend.

Elements and any attributes are entered between matched pairs of angle brackets (< and >), and attribute values are always between a pair of single or double quotes:

```
<element attribute1="value1" attribute2='value2' ...>
```

Entity references start with an ampersand and end with a semicolon:

```
&eref;
```

Element, attribute and entity names are all case-sensitive, so <Element>, <element> and <ELEMENT> are corresponding to three different element types. Likewise, <element att="1" ATT="2> denote two different attribute values for the given element. Naturally, one would be careful with this feature, as misuse could easily confuse users of the tag set.

5.9 Compressing techniques

In this section two ways of reducing the size of an XML document will be briefly explained: Reducing tag redundancy, and binary end-to-end compression.

5.9.1 Tag redundacy reduction

XML files often contain many repetitions of long tag-names, with matching end-tags. With XML namespaces the tags just become even longer. A book list encoded in XML for instance, would probably contain hundreds and hundreds of <AUTHOR>…</AUTHOR>-tags. This naturally leads to XML files taking up much more space than they actually would need to do, but it is actually possible to avoid it. By sacrifising some of XML's human-readable properties, we can replace all similar-looking tags with shorter, non-human-readable tags. For instance, we could replace <AUTHOR> with <A> and still have a valid XML file. WAP Binary XML format (WBXML) is a standard endorsed by the W3C, which helps transform a normal XML document to a tag redundancy reduced WBXML file in a standardised way. WBXML will be used in this project.

5.9.2 Binary compression

Binary compression is well-known from standards such as PKZIP and gZIP. Text-based data can often be compressed up to 90% with binary compression techniques, and that might be well worth a try for a format as text-based as XML. The compression procedure will introduce overehead at both the sending and the receiving end, so tests will have to be performed to see if the compression gain is worth the overhead loss.

5.10 XML parsing

It is stated in research hypothesis **H1** from section 2.5, that we want to minimize the total time taken to process an XML document on the client side. In achieving this, a certain degree of XML parsing will be necessary, both on the optimising proxy server and on the mobile client. In this section we will take a closer look at the field of XML parsing, as there are several parameters to pay attention to, especially when dealing with limited devices.

On the limited, mobile client, XML parsing can be quite a challenge. As discussed in Chapter 3: , some of the limitations on such a device include:

1) The setup time of a network connection is slow.

2) Data transfer rates are slow.

3) Processor speeds are slow.

4) Available memory is limited.

An approach to the first point could be to collect and aggregate messages going to the same client, the second point can be addressed by compacting the XML document to minimize the amount of data transferred. However, there are no solutions directly related to parsing that can fix these problems. On the other hand, point 3 and 4 can be greatly improved by selecting the correct kind of parser for the job.

Several XML parsers with different properties are available for Java today [18], each one tailored for different situations (Table 1). Many of these turn out to be unusable on mobile devices running CLDC because they are too memory dependant, or they utilize classes that are unavailable in the CLDC limited API.

Name	License	Size	MIDP	Type
ASXMLP 020308	Modified BSD	6 kB	yes	push, model
kXML 2.0 alpha	EPL	9 kB	yes	pull
kXML 1.2	EPL	16 kB	yes	pull
MinML 1.7	BSD	14 kB	no	push
NanoXML 1.6.4	zlib/libpng	10 kB	patch	model
TinyXML 0.7	GPL	12 kB	no	model
Xparse-J 1.1	GPL	6 kB	yes	model

Table 1 - A selection of available XML parsers

There are three fundamental kinds of parsers for XML documents. Each kind has its strengths and weaknesses in different situations.

- A *model* parser reads the entire document and produces a structured, object-based representation of the document in the system memory. This way one can freely move around the tree structure of the document an make changes at will. This kind of parser is too heavy weight for use in mobile applications, both in terms of size, memory consumption, and services offered that will not be utilised.

- A *push* parser works its way through the entire document and utilises a system of listeners and events to notify a given object when it encounters different elements in the XML document. This parser implementation is more memory friendly than the latter one, but the mechanism can be tricky to use.

- A *pull* parser is probably the easiest and most basic way of parsing. The entire document is read bit by bit while the controlling application drives the parser forward by requesting the next bit repeatedly. With this implementation there is no need to keep a copy of the model in the system memory, but on the downside one has very limited possibilities of moving back and forth during parsing. Advantages include simplicity and ease of use, as well as a minimal memory footprint. In other words, perfect for use in mobile computing.

5.10.1 kXML 2

Several parsers were considered for use on the mobile client in this project, but the choice fell on kXML 2[2], one of the fastest XML pull parsers for Java. There are several faster parsers out there, but kXML has a strong focus on optimal performance on limited devices.

kXML 2 is built upon a standard, common API for XML pull-parsing that is currently under development, XMLPULL[3]. All parsers implementing XMLPULL can easily be interchanged very easily, only by changing the code line where the parser is instantiated.

kXML 2 is open source, meaning that one can pick whichever components that is necessary at any given time, and include them in the project at will. This is useful on mobile units, to minimise the amount of space occupied.

Another distinct feature of the kXML 2 library is that it includes a WBXML parser as well. Since our tag redundancy reduction functionality will use WBXML, this is a perfect choice of parser for the mobile client.

5.11 XML transformations (XSLT) and XPath

XSLT is a language for transforming XML documents into other XML documents. The technology is well-established, and XSLT processors are widely available for Java. (http://www.w3.org/TR/xslt)

XPath is a language for addressing given parts of an XML document, designed to be used by the XSLT language. (http://www.w3.org/TR/xpath)

The idea is to utilize knowledge of the structure of an XML document (either from DTDs or from XML schemas) to request only the relevant pieces, minimizing both the amount of data transferred to the mobile unit, and the amount of parsing that needs to be done on the unit. This is what will happen:

- The **client** sends an XPath query along with the request for an XML document to the proxy server.

---

[2] kxml website: http://kxml.org
[3] xmlpull website: http://xmlpull.org

- The **proxy** fetches the XML document from the content provider.
- The **proxy** performs the requested XPath query on the XML document, transforming it into a much smaller one containing only the relevant data.
- The **proxy** transfers the reduced XML document to the client.

In addition, the aforementioned compacting techniques can be applied to further reduce the size of the dataset.

```
<?xml version="1.0" encoding="UTF-8"?>

<booklist>
     <book isbn="112344444321">
          <author>Ben Kenobi</author >
          <title>ABC</title>
          <price>123</price>
          <publisher>ACME publishing</publisher>
          <year>2000</year>
     </book >
     <book isbn="212333334433">
          <author>Luke Skywalker</author >
          <title>XML for dummies</title>
          <price>1337</price>
          <publisher>Imperial Books</publisher>
          <year>1994</year>
     </book >
     <book isbn="3453437456">
          <author>Leia Organa</author >
          <title>Foobar</title>
          <price>567</price>
          <publisher>Springer</publisher>
          <year>2002</year>
     </book >

</booklist>
```

Figure 4 - XML document before XPath query

For example, Figure 4 depicts a book list returned by a fictional book directory web service. Let's say we are only interested in the titles and listed prices of the books, so the client supplies the following XPath query to the proxy server:

```
"//title|//price"
```

The proxy will fetch the document and apply the query to it, yielding the document shown in Figure 5.

```
<title>ABC</title>
<price>123</price>
<title>XML for dummies</title>
<price>1337</price>
<title>Foobar</title>
<price>567</price>
```

Figure 5 - Result from XPath query

This is clearly a reduction both in size and complexity, but it is no longer a valid XML document. However, using a similar XPath query in an XSL Transformation we can build a fully functional, reduced XML document as shown in Figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>

<booklist>
     <book>
          <title>ABC</title>
          <price>123</price>
     </book >
     <book>
          <title>XML for dummies</title>
          <price>1337</price>
     </book >
     <book>
          <title>Foobar</title>
          <price>567</price>
     </book >

</booklist>
```

Figure 6 - XSLT transformed document

Part III: Own contribution

Chapter 6: A model of a system for XML optimisation

After surveying the relevant technologies available for my task, it was necessary to find out how some of them would work out in practice. The earlier work [10] on this subject describes a basic model for optimising XML data transfer performance, and some source code for a framework was also available. This framework however, was merely a very basic shell consisting of three parts:

- A very lightweight **mobile client** able to request and receive an XML document from a given source,
- A **content provider** serving a selection of XML documents, and
- A **proxy server**, able to set up and receive connections from the client and provider, but lacking functionality to actually manipulate the XML data in any way.

It was my intention to develop a major extension to this model, primarily focusing on the proxy server, and implementing a selection of the aforementioned ideas for optimisation. Furthermore, the effect of the ideas would be tested and evaluated. This part of the report will describe the work conducted with regards to this prototype system, and show and discuss the results.

A basic model for optimisation of XML documents is described in [10]. In this model, the optimisation is mainly achieved by reducing the size of the XML file, primarily through tag redundancy reduction. In my work I want to extend this model to include size reduction by binary compression techniques, and explore the possibilities of utilizing different parsing methods on the client side to optimise resource usage. The model will also provide means to control the amount of XML data transferred to the client by performing an XSL Transformation specified by the client on the proxy server.

The model described in [10] required the content provider to be aware of the system. In other words, the content providers have to be custom-tailored for use with the system. My model will not suffer from this limitation, as it would be a tremendous advantage to have the system work with existing services.

6.12 Model Requirements

The following two sub chapters will describe general requirements for a model, based on the earlier work and research performed in part I, for optimising the transfer of XML data in a mobile environment.

6.12.1 Requirements regarding XML properties

MR1 *Supporting a diversity of parsers*: As earlier described, the performance of XML parsing depends largely on the implementation of the specific parser. Different situations could require different parsers for optimal performance, and the framework should always use the appropriate parser in any given situation.

MR2 *Information adaptation*: Even with the optimal choice of parser, an XML document might be too big and complex to be handled by the limited memory of the mobile device. The model should provide viable alternatives for such situations, for instance splitting up and handling the document in question in smaller fragments, or downloading only the relevant parts of the document onto the mobile device.

MR3 *Compression and compaction*: A central part of the model will be the capability to minimize the amount of transferred data by using compression techniques. Relevant techniques are tag redundancy reduction, and binary compression.

6.12.2 Requirements regarding the mobile environment

MR4 *Dynamic optimising:* The model should take ever-changing parameters such as memory capacity, available bandwidth, battery capacity, urgency and the complexity of the XML document into consideration and automatically try to find an optimal selection of optimisations for any given situation. This can be achieved by developing a system where heuristic rules are used to determine when to apply each optimisation technique.

MR5 *Support for distribution and heterogeneity:* The system will exist in a mobile environment with a high degree of distribution and heterogeneity, and these factors will have to be taken into consideration.

MR6 *Asynchronous communication:* In a mobile environment, we often see poor connection quality with frequent disconnections and reconnections. This has to be handled transparently by the model, for instance by allowing a download interrupted by connection loss to resume when the connection is restored.

6.13 Model description

Based on the model requirements, a model for optimising XML data transfer and processing in a mobile environment can be devised. Much like the earlier model, the idea is to have two components; a function library on the mobile device, and a proxy server connected both to the mobile device, and to the outside world.

The core of the system resides within the proxy server. This server acts as a portal for mobile units to communicate with any external providers of XML data, which I will refer to as “content providers”. The proxy server’s role will be to receive information requests from mobile clients, and perform necessary data gathering from content providers. The resulting XML data will be processed according to information about the calling client

stored in a profile on the proxy server, in an attempt to optimise transfer and processing times of the data. The processed XML data will then be sent back to the mobile client.

The piece of the system residing on the mobile unit is really an extended XML parser. Layer-wise it will be located just below the application layer, providing a function library to any participating applications. There will be functions for making requests for XML data, and functions for extracting information from the resulting XML documents in the most efficient manner. The idea is that this layer can exist independently from the main application, in the same manner as an ordinary XML parser.

In this model, in contrast to the previously suggested model, the content providers do not need to be aware of the proxy servers' existence. All communication from the mobile clients will pass through the proxy servers, so from the content provider's point of view it looks as if the request is made by the same entity that will receive the reply. This is advantageous, as the content providers will not need to be custom-tailored to the system in any way, and the system is adaptable to any available content providers already in existence, such as web services.

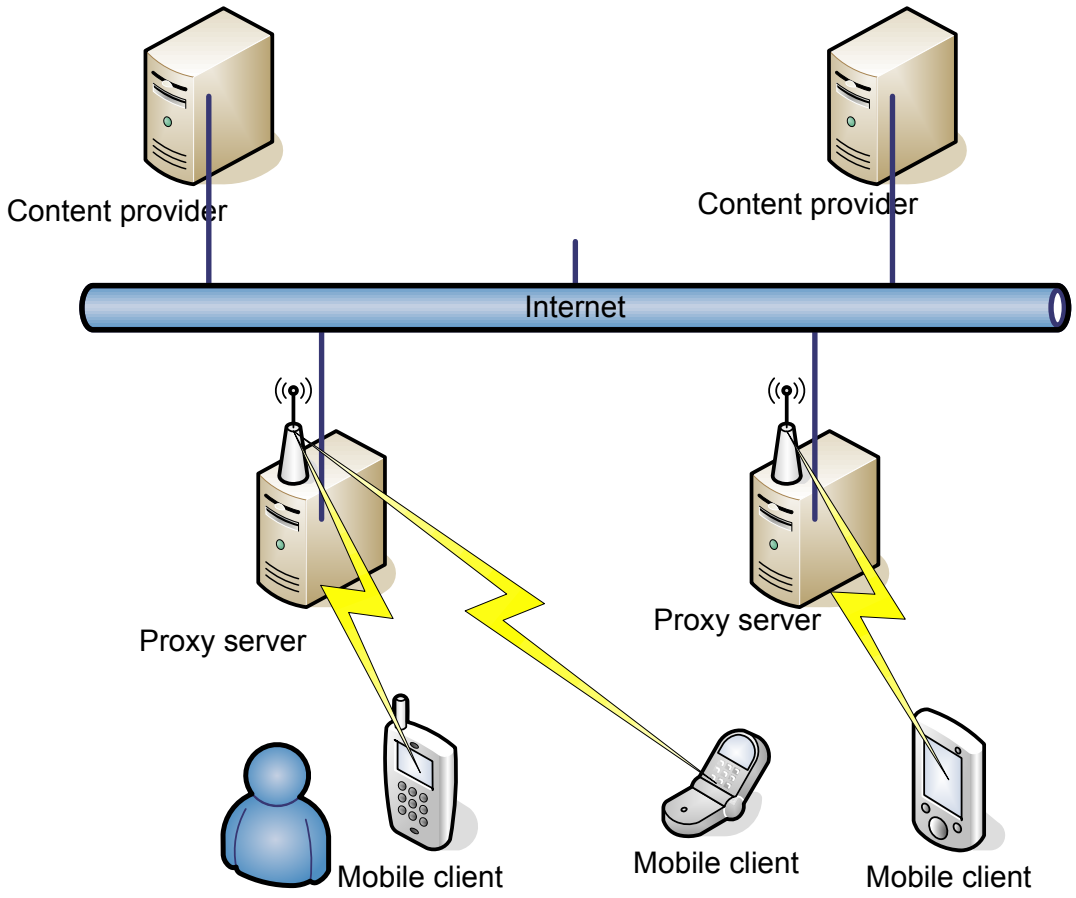

Figure 7 - Overall view of the system

Chapter 7: Requirements engineering

In order to build a system, a requirements specification should be devised. In this chapter we will try to deduce a specification by first constructing a set of use cases for the system.

7.14 Use cases

The intended functionality of the system is presented through a set of use cases [15]. These diagrams are based on the model requirements presented in 6.12, and the earlier work performed in [10]. The primary function of the use cases is to help deduce the requirements, which are presented in the next chapter.

Seven use cases have been created to describe the main functionality of the system. The functionalities described are:

- The client requests an XML document
- The proxy server optimises an XML document
- The content provider returns an XML document to the client through the proxy server
- The client parses a transformed XML document
- Communication is suspended as the client downloads an XML document from the proxy server
- The client uploads an XML document to the content provider through the proxy server

The following three “actors” of the system has been identified:

- The mobile client
- The proxy server
- The content provider

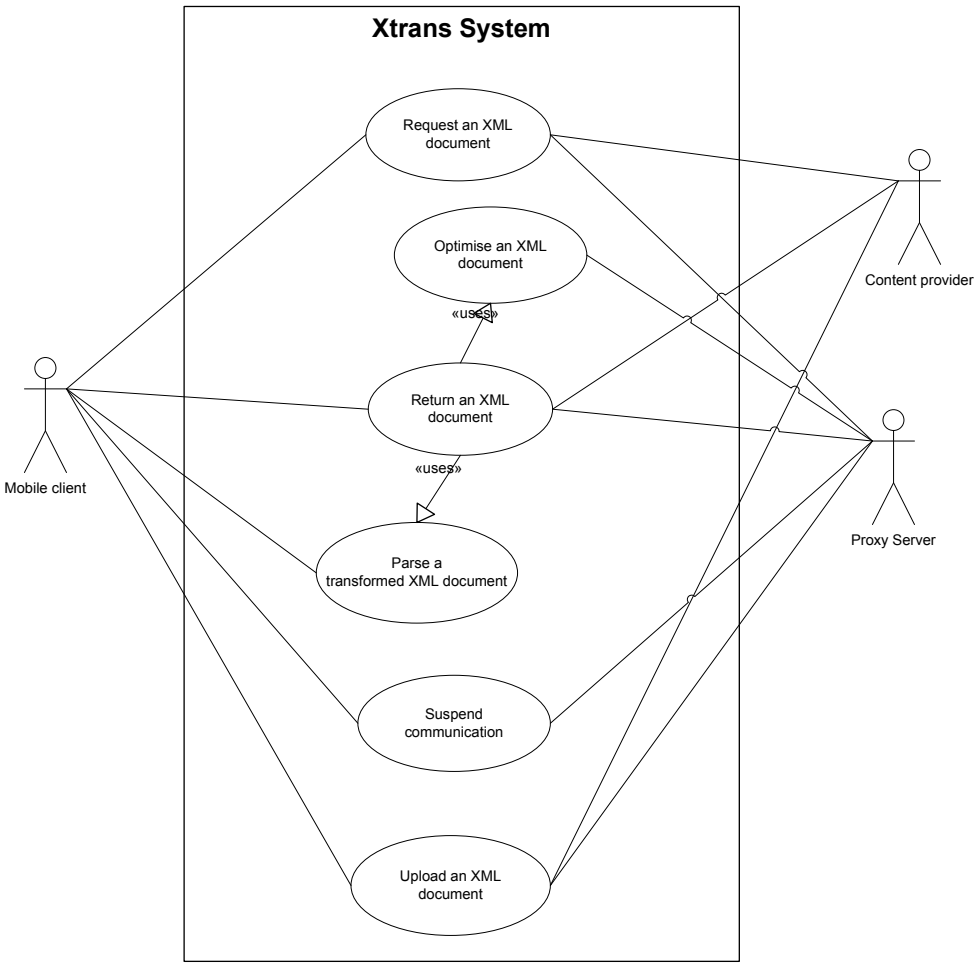

Figure 8 - The XTrans Use Case diagram

The use case diagram shown in Figure 8 shows the six use cases described in this chapter. It also shows the relations between the respective use cases and the actors of the system. Even though the Proxy Server could be considered an integrated part of the system itself and thus should not appear outside the system boundaries, it is shown as an actor in this diagram because it is a good way of illustrating the relationship between the three interacting entities of the system.

Use-case name	Request an XML document
Iteration	Filled
Abstract	The client requests an XML document to be sent from the system.
Trigger	Started by user.
Main path	1: The user starts his application on the mobile client. 2: The mobile client establishes a connection with a proxy server. 3: The mobile client sends its XML document request to the proxy server, along with information about its current state. 4: The proxy server saves the info on the client's current state in its database. 5: The proxy server forwards the request for the XML document to the proxy server.
Alternate path	*Register a new user* 2a: If the mobile client is a new user to the system, it has to register itself in the proxy server. 2b: The mobile client sends information about itself to the proxy server. 2c: The proxy server registers the information in a database. 2d: The proxy-server notifies the mobile client that the system is now ready for use; jump to 3
Exception path	2a: If the connection to the proxy server cannot be established, the user is informed and given a choice whether to cancel or to try again. 2b: If "cancel" is selected the application on the mobile client quits. If "try again" is selected the application tries to establish a connection and if it succeeds; jump to 3.
Preconditions	The user has installed an application on the mobile client supporting the system.
Author	Anders K. H. Walla
Date	27.04.05

Table 2 - Use case #1: Request an XML document

Use-case name	Optimise an XML document
Iteration	Filled
Abstract	An XML document is optimised according to a dynamic set of rules.
Trigger	The proxy server receives an XML document from a content provider.
Main path	1: The XML document is parsed by an XML parser. 2: The structure of the document is analysed and compared to values stored in the rule-base. 3: A decision about how to transform the document is made based on the stored information about the state of the mobile unit and the information about the parsed document. 4: The proxy server applies the relevant transformations to the XML document.
Alternate path	3a: If the documents structure and/or the state of the mobile unit imply no transformation, point 4 is skipped.
Exception path	-
Preconditions	The communication path between the content provider, proxy-server and the mobile client has been initialised.
Author	Anders K. H. Walla
Date	27.04.05

Table 3 - Use case #2: Optimise an XML document

Use-case name	Return an XML document
Iteration	Filled
Abstract	An XML document is sent from the content provider through the proxy server to the mobile client.
Trigger	The content provider receives a request for an XML document from the proxy server.
Main path	1: The content provider looks up the XML document in question. 2: The content provider sends the requested XML document to the proxy server. 3: The proxy-server receives the document from the content provider and handles it as shown in the use-case *"Optimising an XML document"* 4: The proxy-server forwards the document to the destination provided by the content provider. 5: The mobile client receives the XML document from the proxy-server. 6: The document is parsed by the software on the mobile client, as shown in the use case "*Parse a transformed XML document*".
Exception path	-
Preconditions	The communication path between the content provider, proxy-server and the mobile client has been initialised.
Author	Anders K. H. Walla
Date	27.04.05

Table 4 - Use case #3: Return an XML document

Use-case name	Parse a transformed XML document
Iteration	Filled
Abstract	A received XML document is parsed.
Trigger	The mobile client receives an XML document from the proxy server.
Main path	1: It is determined which kind of optimisation techniques are applied to the received XML document 2: The applied techniques are reversed to restore the original XML document. 3: The XML document is parsed by an XML parser. 3: The information from the parsed XML document is provided to the application on the mobile client.
Exception path	-
Preconditions	a) The communication path between the content provider, proxy-server and the mobile client has been initialised.
Author	Anders K. H. Walla
Date	27.04.05

Table 5 - Use case #4: Return an XML document

Use-case name	Suspend communication
Iteration	Filled
Abstract	The connection is broken for some reason during the transfer of an XML document. Both the mobile client and the proxy-server save their current state to be able to resume the transfer at a later stage.
Trigger	The connection between the proxy-server and the mobile client is broken.
Main path	1: The proxy server stops transmitting the XML file. 2: The mobile client saves the received part of the file, and notes how much of the file is missing. 3: The proxy-server notes the point in the XML document when communication was lost and saves the document to a database. 4: The mobile client reconnects to the proxy server and indicates that it wishes to resume the transfer. 5: The proxy server resumes the transfer of the XML file from the given offset.
Alternate path	4a: If the mobile client does not reconnect within a given time (ideally a couple of days), the database entry is removed, and the transfer can no longer be resumed.
Exception path	-
Preconditions	a) The communication path between the content provider, proxy-server and the mobile client has been initialised. b) The proxy-server is sending an XML document to the mobile client.
Author	Anders K. H. Walla
Date	27.04.05

Table 6 - Use case #5: Suspend communication

Use-case name	Upload an XML document
Iteration	Filled
Abstract	An XML document is sent from the mobile client through the proxy-server to a recipient.
Trigger	The user wishes to return an XML document to the content provider.
Main path	1: The application chooses send an XML document. 2: The mobile client transforms the XML document according to the current state of the client. 3: The mobile client includes information about the destination and sends the document to the proxy-server. 4: The proxy-server receives the document from the mobile client and transforms it into a valid XML document. 5: The proxy-server forwards the document to the destination provided by the mobile client. 6: The content provider receives the XML document from the proxy-server.
Alternate path	4a: If the XML is valid upon arrival at the proxy-server, a transformation is unnecessary; jump to 4.
Exception path	-
Preconditions	The communication path between the proxy-server and the mobile client has been initialised.
Author	Anders K. H. Walla
Date	27.04.05

Table 7 - Use case #6: Upload an XML document

7.15 Requirements specification

From the use cases and the underlaying model description, a requirement specification can be deduced. As described in 6.13 the system will have two components: A function library located on the mobile device, and a proxy server for mobile devices to connect to, that will connect to the outside world and optimise the XML data before forwarding it to the mobile device. The functional requirements will therefore be divided into two classes, *client* requirements and *proxy* requirements.

The requirements will be presented according to the pattern defined in [16].

7.15.1 External Interface Requirements

These requirements describe the different kinds of interfaces that the system will provide.

EIR1 *Software Interfaces*: The proxy side will run on the J2SE Virtual Machine, while the client layer is developed in J2ME.

EIR2 *Communications interfaces*: The components will communicate using socket level programming.

7.15.2 Functional Requirements

Functional requirements for the **proxy server**:

PRX-FR1 *Connections:* The proxy server must be able to accept and manage connections from several mobile clients simultaneously. Furthermore, it must be able to establish connections to arbitrary content providers serving XML data on the internet over TCP/IP.

PRX-FR2 *User profiles:* The proxy server has to keep track of its set of users. Each user should have a *profile* stored in the proxy server, that keeps track of the given client's preferences and any interrupted downloads.

PRX-FR3 *XML traffic:* XML documents will be sent and received through the proxy server's connections to the mobile clients, and the content providers. The proxy server needs to know where to route a given document at any time.

PRX-FR4 *Resumed downloads*: If at any time the connection between the proxy server and the mobile client is interrupted while transferring an XML document, the proxy server must be able to resume the data transfer session whenever the connection is restored, with minimal overhead, and no data loss.

PRX-FR5 *XML parsing*: In order to be able to perform XML optimisation, the proxy server needs to be able to parse XML files. The parsing should be performed as efficiently as possible on the given hardware of the proxy server.

PRX-FR6 *XML optimisation*: The overall purpose of the system is to transform an XML document to a form that is more suitable for handling in a mobile environment. This functionality must lie within the proxy server, and includes methods such as redundancy reduction, binary compression, and XSLT transformations.

PRX-FR7 *Dynamic optimisation:* The proxy server will use a set of heuristic rules to determine in which situations to apply the different optimisation techniques to achieve the best performance.

Functional requirements for the **mobile client software**:

CLI-FR1 *Connections:* The client must be able to establish a connection to an available proxy server. It must also provide the possibility to override the optimisation system and connect directly to a content provider.

CLI-FR2 *XML traffic:* The client must be able to both send and receive XML data over an open connection.

CLI-FR3 *Optimised XML parsing:* The client must be able to parse both ordinary XML data, and XML data optimised by the proxy server. This parsing should be conducted in as resource friendly a manner as possible in any given setting.

7.15.3 Performance requirements

The requirements in this section normally state static and dynamic requirements concerning performance, in measurable terms. However, the very purpose of building this prototype system will be to determine actual values for such variables, so specific values will only be stated to a certain extent.

PRFR1 *System performance:* The use of the optimising system must prove an overall gain in performance of at least 10%.

PRFR2 *Load management:* The system must not stop yielding performance increases even though several users are using the system at the same time.

7.15.4 Design constraints

Design constraints may result from such things as the prescribed use of certain standards or hardware.

DSGN1 *Client size:* The maximum size of the software on the mobile application should not exceed 60kB.

DSGN2 *XML adherence:* The standard rules of the XML format should not be "broken" in order to yield a performance increase.

7.15.5 Software system attributes

This section deals with software quality concerns.

SWSA1 *XML restoration:* XML data that is optimised by the system must be able to be fully restored to its original state, even in the case of connectivity loss.

SWSA2 *Source code quality:* The source code must be thoroughly commented and explained so that future enhancements or studies of the software can be conducted.

SWSA3 *Modularity:* The software must be module-based and should utilise well-defined interfaces where possible, so that certain components may be swapped out at a later stage.

7.16 Requirements summary

The limited time and resources of this project unfortunately limits the amount of functionality that can be implemented in the prototype. Table X will prioritise the presented requirements using the values H (high), M (medium) and L (low). The purpose of stating these priorities is to sort out which parts of the system is necessary in order to conduct a test that compares optimisation techniques in different situations, in an attempt to establish a basis for a heuristic rule set. The table also shows which requirements cannot be fulfilled in this iteration due to time constraints.

ID	Description	Priority	Postponed
EIR1	Software interfaces	H	
EIR2	Communications interfaces	H	
PRX-FR1	Connections	H	
PRX-FR2	User profiles	M	X
PRX-FR3	XML traffic	H	
PRX-FR4	Resumed downloads	M	X
PRX-FR5	XML parsing	H	
PRX-FR6	XML optimisation	H	
PRX-FR7	Dynamic optimisation	L	X
CLI-FR1	Connections	H	
CLI-FR2	XML traffic	H	
CLI-FR3	Optimised XML parsing	H	
PRFR1	System performance	M	
PRFR2	Load management	L	X
DSGN1	Client size	M	
DSGN2	XML adherence	H	
SWSA1	XML restoration	H	
SWSA2	Source code quality	H	
SWSA3	Modularity	H	

Table 8 - Prioritised requirements

Chapter 8: Implementation

This chapter will describe the actual implementation of a system that is able to perform XML optimisation, according to the selection of requirements shown in 7.16. The first section will give a general description of the system, while the subsequent sections describe its individual components.

8.17 General description

The system has been described to consist of two main components: A proxy server performing the optimising XML transformations and a layer on the mobile unit able to parse the transformed XML document. However, in order to run the system we need a third component, namely a content provider able to feed XML documents into the system. This could be an external provider, but for testing purposes it is advantageous to have a custom tailored content provider.

The purpose of implementing the prototype at this stage was twofold:

- To evolve the earlier very basic attempt of an implementation into a functional platform for further experimenting and developing.
- To be able to conduct performance testing on some optimisation techniques.

Figure 9 gives an overview of the system architecture. The shaded blocks are considered parts of the system, and the raised boxes are the parts that actually need to be implemented to have a functioning prototype. In the following sections the individual components is described.

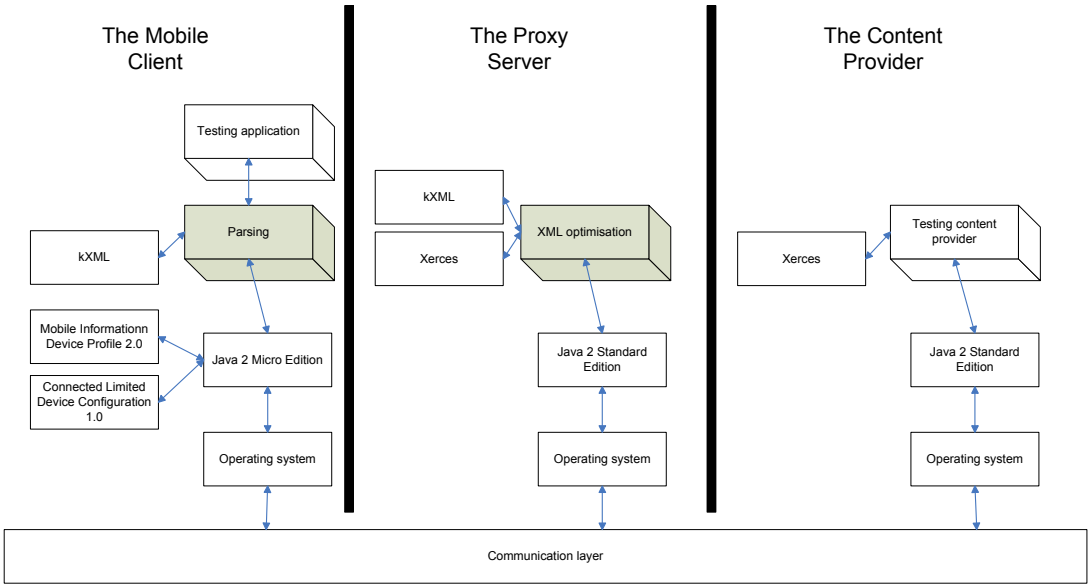

Figure 9 - The overall prototype system architecture

8.18 The communications protocol

A vital part of a system of this kind is the communication between the components. An overall goal is to keep the amount of traffic as low as possible, while still being practical and realistic about message formats. Generally, messages that require metadata to be "piggybacked" use a custom tailored XML message for this purpose. In the following, I will describe a typical scenario with a mobile client requesting a document from the content provider via the proxy server, and the messaging format chosen for the prototype implementation.

The first message to move through the system is the one going from the mobile client to the proxy server. This is originally the message that would go directly to the content provider, but as it must pass through the proxy server it needs to have additional information attached. A sample message is shown in Figure 10.

```
<?xml version="1.0" encoding="UTF-8"?>

<request>
     <header>
          <date>10/07/2005</date>
          <target ip="129.241.102.194" port="8189"/>
          <optimise>2</optimise>
     </header>
     <body>
          marc100.xml
     </body>
</request>
```

Figure 10 - Sample message from client to proxy

This message has information on where the date of issue, where the message is to be forwarded (the content provider's address) and what kind of optimisation should be performed (we will look at the enumeration for this later). The <body> section contains the part that will actually be sent to the content provider. In this instance it is merely the file name of the XML document in question, but in cases where the content provider is a web service provider it can be another XML document nested within the request. Information such as the client's own IP address is not necessary at this stage, as the communications channel with the proxy server will be kept open during the operation. In a future implementation however, this might be implemented as part of the profile functionality described in **PRX-FR2**. Any information about the state of the client will also be located in the <header> section in later iterations.

The proxy server now knows where to forward the given message. The next message will be the one going from the proxy server to the content provider. In our case the content provider simply returns a named XML document stored on the provider server, and it accepts messages such as the one shown in Figure 11.

```
marc100.xml
```

Figure 11 - Sample message from proxy to content provider

Our content provider's reply to this message will be the XML document in question. A sample document is shown in Appendix A . This document is sent directly back to the proxy server through the established communications channel.

After being processed by the proxy server, the now optimised XML document is finally sent unwrapped back to the awaiting client. At a later stage, when the proxy server might have a larger degree of control of when to perform different optimisations it might be necessary to wrap this optimised XML document in another XML document in order to supply information about what optimisations that have been performed and will need to be reversed on the client side.

An overview of the messages exchanged between the three actors is shown in Figure 12, which also shows when the respective components are active.

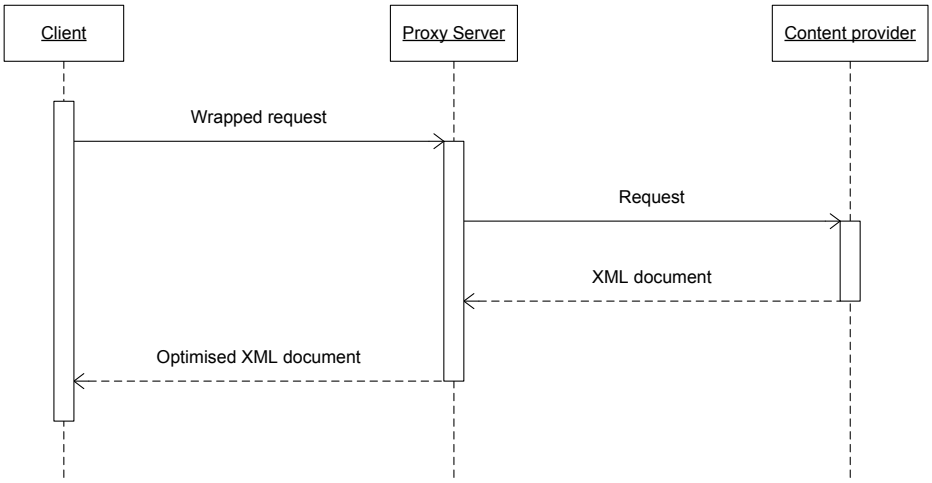

Figure 12 - UML Sequence diagram showing messaging

8.19 The Mobile Client

The model describes the software on the client as a "layer" of functions to be offered to any mobile application. The reason for having a part of the system on the mobile unit is naturally to "reverse" any effect the optimisation performed by the proxy server have inflicted on the XML. An approach to this would be to fully restore the XML document to its original state before offering it to the application as a stream of XML data, as shown in Figure 13. This will require the mobile application to take care of the XML parsing on its own, but it will be a clear and easy way to separate the system from the application.

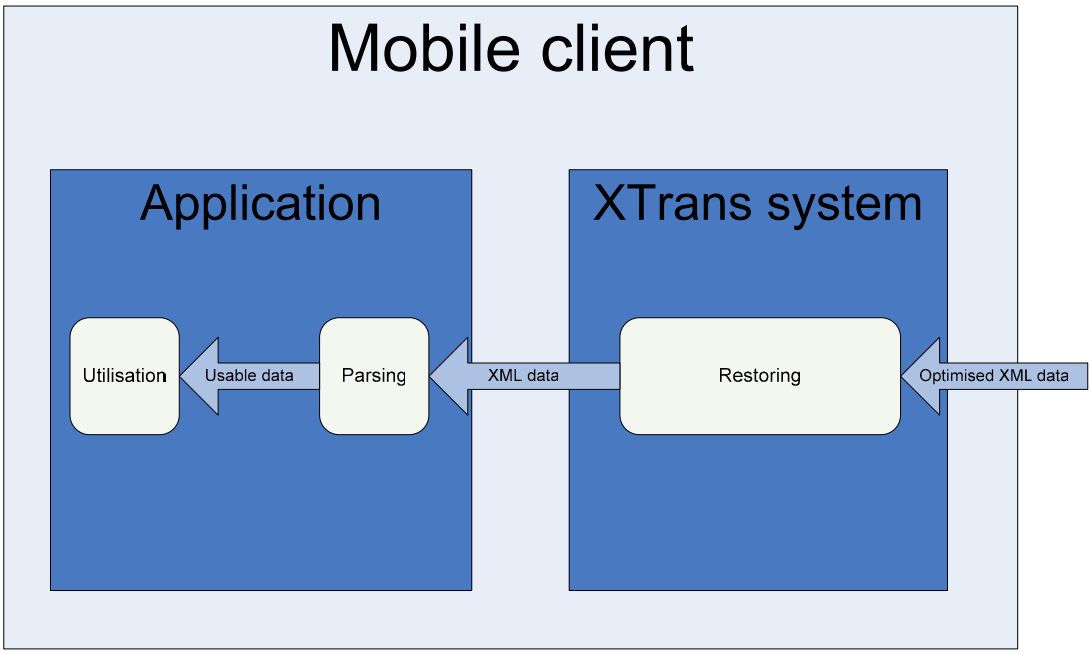

Figure 13 - Data flow within the client, option A

Another approach (shown in Figure 14) would be to include the XML parsing in the XTrans layer on the mobile device. This allows us to a) ensure that the parsing is done in the most optimal way possible with regards to choosing the best parser in a given situation, and b) to combine restoring and parsing the document, when this is possible. This will often eliminate one of the iterations through the dataset and thus increase performance. However, this approach imposes another challenge on us: How to consistently present the parsed data to the application in an orderly manner. An option would be to supply a typical parser's interface between the system and the application, such as the XmlPullParser[4], but this would not be very elegant and may be too hard to enforce efficiently in practice.

[4] See http://www.xmlpull.org/

Deciding on and implementing one of the discussed approaches is left for further work, and the boundary between the application and the system remains somewhat clouded in this implementation.

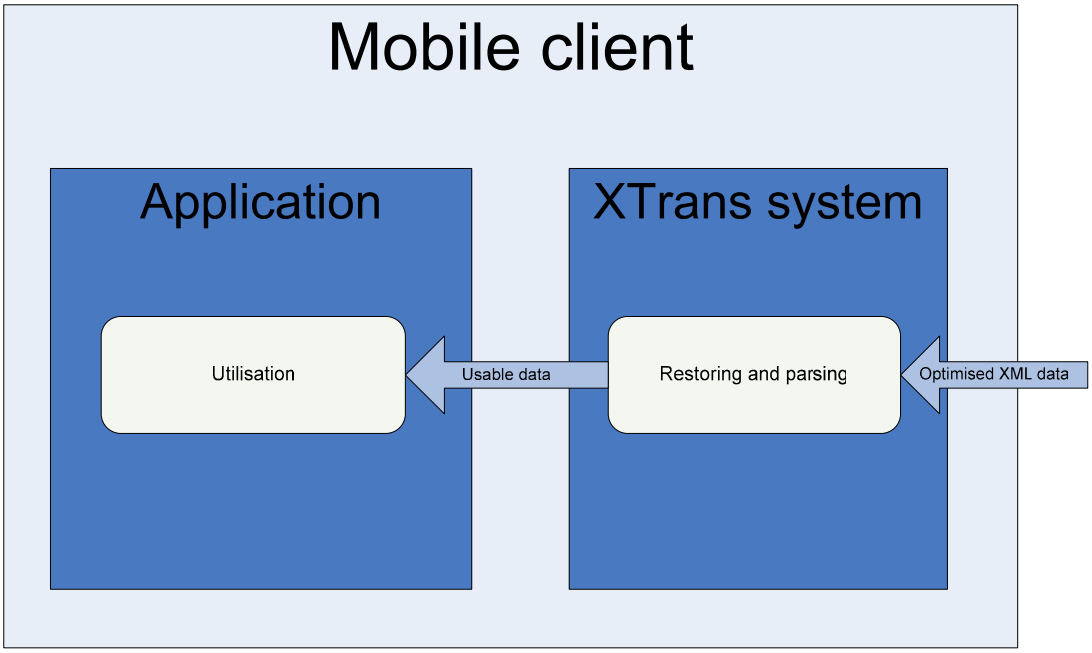

Figure 14 - Data flow within the client, option B

The implemented client is a basic application that asks for some user input regarding the request to be made through a simple interface. On the arrival of the requested document, the different stages of the XML restoration are timed and presented. The client performs a typical parse operation on the received XML book data.

8.20 The Proxy Server

The proxy server implementation is fairly straight forward. The main thread runs a ServerSocket that constantly listens for incoming connections. Whenever a connection attempt is detected, a separate thread, called a SocketConnection, is spawned to handle that specific client. The connection to this client is kept alive during the entire transaction. By using separate independent threads to handle the incoming connections, the server is capable of handling multiple requests simultaneously, which is a natural requirement of such a system.

The SocketConnection will receive the request from the client and separate the encasing metadata from the request aimed for the content provider. Information regarding what optimisation techniques to apply is included in the metadata, and is saved in the thread for later use. Once the information about the content provider is extracted, the proxy server establishes a connection and forwards the request.

When the request is sent, the proxy server will wait for the resulting XML document to arrive from the content provider. On arrival the document will be parsed and analysed according to the clients request for optimisation, and the requested optimisation techniques will be applied. The available techniques in this implementation are binary compression, tag redundancy reduction, or a combination of both. The optimisation steps are timed, and a summary of the time consumed is shown at completion.

Finally the optimised data is streamed on to the client over the waiting, open connection. Any engaged resources are released and the thread is closed.

8.21 The Content Provider

The content provider implementation is a static one, mainly functioning as a placeholder while the rest of the system undergoes a few iterations. At a later stage, “real” content providers should be supported, such as web service providers, but while the system is in its early stages of development a custom-tailored one is easier to deal with.

The service provided is very simple: The content provider contains a database of static XML files, and these can be accessed by establishing a socket connection and simply sending the name of the file. The files provided in this implementation are:

- marc1.xml (1.46 KB) – A small document containing one record of a booklist.
- marc100.xml (37.1 KB) – A medium sized document containing a hundred records of a booklist.
- marc300.xml (111 KB) – A relatively large sized document containing 300 records of a booklist.

The requested file will be returned to the proxy server over the same socket connection.

Chapter 9: Testing

In order to find an answer to research hypothesis **H1** presented in section 2.5, I found it necessary to evaluate some optimisation strategies in different situations to try to determine parameters for use in future work on this matter. From earlier experiences [17] I knew that there was a lot of potential in compressing and compacting XML data for transfer in mobile environment, just not to which extent.

9.22 Planning

The main purpose of these test cases is to compare the two main compression techniques explored in this report: Binary compression, in our case a gZIP library, and tag redundancy reduction, for which we use the WBXML standard. Both are explored in the state of the art survey. In addition to comparing the techniques to one another, both techniques will be applied simultaneously, and will also examine the case of no compression, for comparison purposes.

The testing will be performed by using the prototype system which was described in the earlier chapters. The code on the mobile device and the proxy server has been modified so that it will display the amount of time spent on different tasks related to the techniques at hand. It will also reveal any gains or reductions in file size.

The XML data used for the testing is a book list provided by Bibsys' SRU client, which is freely available on the web[5]. A sample is shown in Appendix A. To fully explore the effect that overhead may have on differently sized data, three tests will be conducted: A small document containing one record, a larger document containing 100 records, and a fairly large document containing 300 records. Each technique is performed three times on every document, and the results are averaged, to minimise the effect of any deviations.

Unfortunately, a mobile phone capable of running the designed prototype software was not available at the time of the testing. The content provider and the proxy server were run on different ports on the same computer, and the client was run on an emulator, also on the same computer. This means that issues related to bandwidth will be hard to spot, but it would illuminate non-bandwidth related issues.

[5] See: http://www.bibsys.no/z/sru.html

9.23 Execution

The results from the testing are presented in the following tables. All the times are given in milliseconds.

9.23.1 Small document

Small		Size		Time on proxy						Time on client					
Method	Iteration	Bytes	%	Compressing	Tag reduction	Transferring	Overhead	Total	%	Transferring	Parsing	Decompressing	Overhead	Total	%
Plain XML	1							0		16	218		16	250	
	2							0		16	266		31	313	
	3							0		15	219		16	250	
	AVG	1497	100 %							16	234		21	271	100 %
Tag reduction	1				62	0	0	62		15	78		16	109	
	2				141	109	0	250		109	78		32	219	
	3				125	109	0	234		110	63		15	188	
	AVG	1243	83 %		109	73	0	182		78	73		21	172	63 %
Compression	1			0		109	16	125		109	172	47	47	375	
	2			0		109	16	125		110	187	62	16	375	
	3			0		0	0	0		16	203	31	16	266	
	AVG	619	41 %	0		73	11	83		78	187	47	26	339	125 %
Tag reduction and compression	1			0	31	109	0	140		109	63	47	15	234	
	2			0	0	109	0	109		110	78	31	16	235	
	3			0	31	0	0	31		16	110	31	15	172	
	AVG	655	44 %	0	21	73	0	93		78	84	36	15	214	79 %

Table 9 - Test case 1: Small document

Note that the Time on proxy column does not have percentage values. This is because the time spent by the proxy forwarding the small document came out as 0, which is pointless to compare to.

9.23.2 Medium document

Medium		Size		Time on proxy						Time on client					
Method	Iteration	Bytes	%	Compressing	Tag reduction	Transferring	Overhead	Total	%	Transferring	Parsing	Decompressing	Overhead	Total	%
Plain XML	1							187		344	2672		16	3032	
	2							62		344	2797		15	3156	
	3							171		375	3001		15	3391	
	AVG	38059	100 %					140	100 %	354	2823		15	3193	100 %
Tag reduction	1				157	109	0	266		188	485		15	688	
	2				78	109	0	187		187	531		0	718	
	3				47	109	0	156		188	750		15	953	
	AVG	19308	51 %		94	109	0	203	145 %	188	589		10	786	25 %
Compression	1			0		109	32	141		110	2672	343	16	3141	
	2			0		109	32	141		94	2688	281	0	3063	
	3			16		109	16	141		109	2844	453	16	3422	
	AVG	5829	15 %	5,3		109	27	141	101 %	104	2735	359	11	3209	100 %
Tag reduction and compression	1			0	31	109	0	140		110	422	312	16	860	
	2			0	62	0	0	62		47	407	281	15	750	
	3			0	32	109	0	141		110	421	266	31	828	
	AVG	5520	15 %	0	42	73	0	114	82 %	89	417	286	21	813	25 %

Table 10 - Test case 2: Medium document

9.23.3 Large document

Large		Size		Time on proxy						Time on client					
Method	Iteration	Bytes	%	Compressing	Tag reduction	Transferring	Overhead	Total	%	Transferring	Parsing	Decompressing	Overhead	Total	%
Plain XML	1							812		984	7312		16	8312	
	2							735		906	7187		0	8093	
	3							734		906	7187		32	8125	
	AVG	110737	100 %					760	100 %	932	7229		16	8177	100 %
Tag reduction	1				172	328	0	500		437	1438		16	1891	
	2				172	343	0	515		468	1500		16	1984	
	3				62	344	0	406		469	1484		0	1953	
	AVG	54947	50 %		135	338	0	474	62 %	458	1474		11	1943	24 %
Compression	1			15		0	157	172		141	7266	468	32	7907	
	2			15		110	63	188		141	7282	578	30	8031	
	3			0		125	94	219		141	7156	750	31	8078	
	AVG	14446	13 %	10		78	105	193	25 %	141	7235	599	31	8005	98 %
Tag reduction and compression	1			16	78	125	0	219		141	1469	594	15	2219	
	2			16	47	109	0	172		125	1469	500	16	2110	
	3			16	63	125	0	204		141	1469	468	32	2110	
	AVG	13342	12 %	16	63	120	0	198	26 %	136	1469	521	21	2146	26 %

Table 11 - Test case 3: Large document

9.24 Result analysis

The test cases reveal several interesting, positive results. The most apparent one is that the binary compression truly excels at limiting the space consumed, reducing a medium or large sized document to about 15% of it's original size. Also, the compression procedure is a trivial task for a server-sized computer, and is performed at a maximum of 16ms, even on the largest document. On the mobile client the decompression take a little longer to perform, but still at a very low addition in time. On the smaller documents, there is a small time penalty to using binary compression; at larger sizes this penalty is insignificant.

The tag reduction scheme performs adequately at compressing data, averaging at about 50% for medium/large documents. However, the most interesting point on this technique is the tremendous increase in parsing speed. This is most likely due to the fact that the tag reduction library used does not transform the tag reduced XML file back to its original form in order to parse it, but rather sticks with the reduced tags. This saves the mobile client from the need to perform as many string compare operations, which are relatively resource heavy operations. The tag reduction scheme yields a speed increase of roughly 75% on medium and large sized documents.

An interesting point to note is that the combination of the two techniques provides a highly desirable performance increase combined with size reduction, losing only a few percentages to the optimal performance with either technique.

As a side note, looking at the time usage on the proxy server shows that the non-compressed data usually yields the longest processing times. A closer look reveals that the time consumed on the proxy is proportional to the size of the resulting XML document, and that indicates that transferring the file is what takes up the most time, not processing the data. This is good news because there does not appear to be a heavy demand for processing power on the proxy server, especially since the proxy server will most likely be handling several clients at once.

It is also intriguing to note that we here have two different techniques that yield two different kinds of performance increase, speed and size. They are therefore perfect targets for the planned heuristic rule set. The binary compression technique can be applied when the bandwidth is limited, while the tag redundancy reduction scheme can be applied when time, or low battery consumption is of the essence. However, according to these tests it would appear that the optimal solution at any time would be to use both tag redundancy reduction and binary compression, as this gives the best overall increase in size and speed.

It should be noted that the tests were run on a single computer, and that running the same tests in a distributed environment could illuminate different issues.

Part IV: Discussion and conclusion

Chapter 10: Discussion

This report has looked into several issues regarding optimisation of transfer and processing of XML-based information between mobile, heterogeneous clients and supporting servers. A survey of the current state was performed, and through a requirements engineering process, a model was built which aims to fulfil these requirements.

The model has also been put to life through a working implementation, based on technologies from the state-of-the-art studies. Two optimisation schemes have been thoroughly tested, and found to be satisfactory effective, both in different, complimentary ways. The testing was however performed on a single, stationary computer, so the results could possibly be slightly different in a distributed environment.

Research hypothesis **H1** was tested and found to be true, there are definitely techniques available to optimise XML traffic in mobile environments. Hypothesis **H2** has been thoroughly looked into, and while no verification has been conducted, the hypothesis has been significantly strengthened by the two implemented optimisation schemes, as they have proved to be viable candidates for use in a heuristic rule-set:

The tag reducing scheme was found to not only reduce the size of a document, but more importantly it increased the client-side parsing speed by a factor of 4.5x. This is a very significant increase, especially considering that the parsing is the most time-consuming activity involved in receiving an XML document, according to the tests. Encoding files for use with the tag reducing scheme on the proxy server was also shown to be quite a trivial task, as the proxy server has more than enough resources available.

The binary compression on the other hand, proved to be highly effective at size-reduction, reducing the larger documents in the test by up to 85% in size, with only a reasonably small performance hit on the client side when decompressing. Some may argue that applying binary compression to the data removes the properties of XML data while the data is in transit, and should therefore be avoided. On the other hand, the transformed tag reduced data is also close to unusable while in transit unless one has prior knowledge of the decoding scheme.

However, what might weaken **H2** is the fact that when both tag reducing and binary compression are combined we actually get the best of both worlds: near-optimal size-reduction and speed-increase. In practice this means that as long as the file size is above a given threshold we should simply always apply both techniques and have optimal performance, and there will be no use for the system proposed in **H2**.

Chapter 11: Future work

Designing a complete framework for XML traffic optimisation is not a trivial task. My work however, shows a large step towards a complete solution, and gives important test data for further development. It is my belief that a complete system for optimising XML data handling in a mobile environment could be developed within a few semesters. The following section outlines steps that should be taken towards this goal.

The supplied implementation is a prototype in its early stages. However, the basis has been made for a larger and richer prototype that could eventually lead to the realisation of this system.

An immediate next step would be to implement XSLT transformations on the proxy server, requested by the client, and conduct performance testing on this. However, this was the first feature that had unfortunately had to be left out of this implementation due to time constraints, as it is fairly obvious that cutting out only the necessary bits of the XML document will lead to a significant decrease in file size. The time aspect of this on the other hand, is most interesting. Implementation and testing will show whether the time spent on the transformation is worth the size reduction, also in relation to and combined with the other techniques.

Further testing should be conducted in a distributed setting. The content provider and the proxy server should run on different servers, and the mobile client should, if possible, be run on an actual mobile unit. This will ensure more reliable and useful test data, and is imperative as a basis for developing the heuristic rule set.

The self-defined content provider should eventually be replaced with actual content providers, such as web service providers.

When the aforementioned measures have been taken and enough data has been gathered, attention should be paid to the defining of the heuristic rule set. With the implementation of this feature, the system will be able to dynamically scale itself to different situations and conditions.

A solution to the layering problem discussed in 8.19 should be sought to provide a practical plug-in solution for mobile applications on the client side.

Chapter 12: Conclusion

This report has described a prototype middleware system for optimising transfer and processing times of XML based data between mobile, heterogeneous clients, supporting servers and content providers. Firstly, a state-of-the-art survey was conducted, in order to uncover necessary and available technologies for such a solution. Then a model was built, attempting to cover relevant details of a solution, and a requirements specification was set up.

A prototype system has been implemented, according to a selection of the requirements. In this system, a client can request an XML file from a content provider through a proxy server. The proxy server will fetch the document from the content provider, and perform a selection of optimisation techniques on this XML document before sending it back to the client, who parses the document. Two such optimisation techniques have been implemented: Tag redundancy reduction and binary compression.

The two techniques was thoroughly tested with documents ranging up to about 100KB, and found to be satisfactory effective. The tag redundancy scheme managed to reduce the parsing time on the mobile client significantly, and the binary compression scheme heavily reduced the size of the document. The techniques were also proven to work in conjunction with each other, yielding the best benefits of both techniques.

Placed in the context of having a heuristic set of rules determining when to apply a given optimisation technique in a given situation, these two schemes fit right in, as each one has its speciality. Tag reducing could be used when fast or battery-friendly parsing was required, and binary compression could be used when the bandwidth was low. However, the fact that both schemes paired together seem to yield the most optimal solution for all documents over a given size seem to defeat the purpose of having a dynamic selection of optimisation techniques. More thorough testing in actual distributed environments should be conducted to gain more insight into this question.

References

[1] http://en.wikipedia.org/wiki/XML [Last visit: 01/03/05]

[2] http://www.w3.org/TR/2004/REC-xml-20040204/ [Last visit: 01/03/05]

[3] Michel Goossens and Sebastian Rahtz. *The Latex Web Companion*. Addison Wesley, 1999: Chapter 6: *HTML, SGML, and XML: Three markup languages*

[4] http://java.sun.com/j2me/ [Last visit: 10/06/05]

[5] Sun Microsystems. *J2ME Datasheet*. (URL: http://java.sun.com/j2me/j2me-ds.pdf) [Last visit: 21/04/2004]

[6] Helal, Sumi. *Pervasive Java Part 1.* Pervasive Computing, 1536-1268/02, 2002

[7] Helal, Sumi. *Pervasive Java Part 2.* Pervasive Computing, 1536-1268/02, 2002

[8] M. Berndtsson, J. Hansson, B. Olsson and B. Lundell. *Planning and Implementing your Final Year Project with Success!* Springer, 2002

[9] Knutsen, Andreas. *Cheaper parsing of XML on Mobile Devices*. NTNU - Autumn 2003

[10] Melcher, Tobias. MOWAHS – Optimised Xml Management in Mobile Environments. NTNU - Autumn 2004

[11] George H. Forman and John Zahorjan. *The Challenges of Mobile Computing*. April 1994.

[12] van Vliet, Hans. Software Engineering, Principles and Practice, Second edition. John Wiley & Sons Ltd, 2000

[13] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12, 1990.

[14] http://www.mowahs.com/ [Last visit: 01/06/05]

[15] Martin Fowler, *UML Distilled – Second edition*. Addison-Wesley, 2002

[16] IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1993.

[17] Walla, Anders and Berg, Morten, *Mobilt biblioteksystem*, NTNU - Autumn 2004

[18] Knudsen, Jonathan, *Parsing XML in J2ME,* Sun Microsystems, (URL: http://developers.sun.com/techtopics/mobility/midp/articles/parsingxml/) 07/03/02, [Last visit: 01/05/05]

[19] Mark Weiser, *The Computer for the Twenty-First Century*, Scientific American, pp. 94-10, September 1991

Appendix A XML sample

A sample XML file as provided by the content provider.

```
- <SRW:searchRetrieveResponse xmlns:SRW="http://www.loc.gov/zing/srw/v1.0/"
    xmlns:DIAG="http://www.loc.gov/zing/srw/v1.0/diagnostic/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcx="http://krait.kb.nl/coop/tel/handbook/telterms.html"
    xmlns:tel="http://krait.kb.nl/coop/tel/handbook/telterms.html"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mods="http://www.loc.gov/mods"
    xmlns:lib="http://krait.kb.nl/coop/tel/handbook/telterms.html"
    xmlns:cld="http://www.ukoln.ac.uk/metadata/rslp/schema/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <SRW:numberOfRecords>779</SRW:numberOfRecords>
    <SRW:resultSetId>20050625.713.kqf</SRW:resultSetId>
  - <SRW:records>
    - <SRW:record>
        <SRW:recordSchema>kb_caption</SRW:recordSchema>
      - <SRW:recordXML>
          <db-code>EWTIJ</db-code>
          <title>Dynamically Updating XML Data: Numbering Scheme
            Revisited</title>
          <author>Jeffrey Xu Yu</author>
          <year>2005</year>
          <volume>8</volume>
          <issue>1</issue>
          <pages-range>5-26</pages-range>
          <type>article</type>
        </SRW:recordXML>
        <SRW:recordPosition>1</SRW:recordPosition>
      </SRW:record>
    </SRW:records>
  - <SRW:diagnostics>
    - <DIAG:diagnostic>
        <DIAG:code>-1</DIAG:code>
        <DIAG:details>Unknown schema</DIAG:details>
      </DIAG:diagnostic>
    </SRW:diagnostics>
  - <SRW:echoedRequest>
      <SRW:query>XML</SRW:query>
      <SRW:maximumRecords>1</SRW:maximumRecords>
      <SRW:recordSchema>http://www.loc.gov/marcxml/</SRW:recordSchema>
      <SRW:startRecord>1</SRW:startRecord>
    </SRW:echoedRequest>
  </SRW:searchRetrieveResponse>
```

Appendix B Javadoc

Javadoc for the XTrans-system can be found on the enclosed CD, along with the source code.