Andreas Knudsen

Cheaper parsing of XML on Mobile Devices

A Project in the course
TDT4735 Systemutvikling, fordypningsemne

Faglærer: Alf Inge Wang
Veileder: Carl-Fredrik Sørensen

Institutt for datateknikk og informasjonsvitenskap, NTNU
Høsten 2003

1 Abstract

As more and more systems opt for platform independence and interoperability, XML is selected by many as the data format of choice. XML features a standard way of representing data which can be utilized by a multitude of different systems.
In addition, XML guarantees forward compatibility, which gives developers more confidence that their products will last past the next update.

The fact that using XML is a rather resource intensive activity usually only becomes an issue when it is to be used by resource-poor devices such as handheld PDAs or mobile phones. To effectively use large XML files for these devices, something needs to be done as these resource-poor devices still need to be able to network with larger and stronger devices such as desktop computers and servers.

To find out what can be done, it is important to know which properties in XML regulate its expensiveness. This report is aimed at finding such properties and identifying the consequences of varying these to parsing of XML by weak devices. In addition, a hypothesis that tag-redundancy can be exploited to make parsing of XML cheaper is tested.

2 Preface

This report is the result of work performed in a project at the Department of Computer and Information Science, Norwegian University of Science and Technology, during the autumn of 2003. The project is a part of the education for the degree of "Master of Technology: Computer Science" in the ninth semester in the course "TDT4735 Fordypningsemne, Systemutvikling".

I would like to extend my gratitude to two persons: The first is Carl-Fredrik Sørensen, at the Department of Computer and Information Science, NTNU, who has been my supervisor during the project. Though he has been in England for most of the project he has been of great help. The second is Alf Inge Wang, also within the Department of Computer and Information Science, who also assisted me throughout the project.

3 Table Of Contents

1 ABSTRACT 2

2 PREFACE 3

3 TABLE OF CONTENTS 4

4 LIST OF FIGURES 6

5 LIST OF TABLES 7

6 INTRODUCTION 8

7 PRESTUDY 10

8 RESEARCH AGENDA 19

9 OWN CONTRIBUTION 21

9.1 FINDING PARAMETERS TO TEST: 21
9.1.1 LITERATURE STUDY: 21
9.1.2 EDUCATED GUESSES 21
9.2 EXPERIMENT 23
9.2.1 EXPERIMENT DEFINITION: 23
9.2.2 EXPERIMENT PLANNING 24
9.2.3 EXPERIMENT OPERATION: 29
9.2.4 EXPERIMENT ANALYSIS & INTERPRETATION 30
9.2.5 ANALYSIS AND INTERPRETATION: 39

10 RESULTS 44

11 EVALUATION 45

11.1 SIGNIFICANCE OF RESULTS: 45
11.2 VALIDITY OF RESULTS: 46

12 CONCLUSION 47

13 FURTHER WORK: 48

14 INDEX: 49

15 BIBLIOGRAPHY 50

APPENDIX A 52

APPENDIX B 55

B.1 XML USED IN TESTING HYPOTHESIS A. 55
B.2 ORIGINAL XML USED IN TESTING HYPOTHESIS D. 56
B.3 REDUCED XML USED IN TESTING HYPOTHESIS D. 56

APPENDIX C 57

APPENDIX D 59

4 List of Figures

Figure 7-1: Overview of the Java technology. [17]……..…13

Figure 7-2: Overview of Java 2 Micro Edition. [17]……..……14

Figure 7-3: Conceptual model of tag reducing system, server side…….……..15

Figure 7-4: Example of tag reduced XML…….……17

Figure 7-5: Conceptual model of tag reducing system, client side…….……...18

Figure 9.2-1: Overview of test system…….……..29

Figure 9.2-2: Results from testing hypothesis 1A…….30

Figure 9.2-3: Results from testing hypothesis 2A…….31

Figure 9.2-4: Results from testing hypothesis 1B..……...32

Figure 9.2-5: Results from testing hypothesis 2B. ……..33

Figure 9.2-6: Results from testing hypothesis 1C. ……..34

Figure 9.2-7: Results from testing hypothesis 2C. ……..35

Figure 9.2-8: Results from testing hypothesis 1D. ……..36

Figure 9.2-9: Results from testing hypothesis 2D. ……..37

5 List of Tables

Table 9.2-1: Project definition. ……....23

Table 9.2-2: Alternative hypotheses……..24

Table 9.2-3: Effect on comm. cost when reducing tags……....38

Table 9.2-4: T-test values for hypothesis 1A…….40

Table 9.2-5: T-test values for hypothesis 1B…….40

Table 9.2-6: T-test values for hypothesis 2A…….41

Table 9.2-7: T-test values for hypothesis 2B…….41

Table 9.2-8: T-test values for hypothesis 1D…….42

Table 9.2-9: T-test values for hypothesis 2D…….43

6 Introduction

This project was selected because I have a great interest in networking with mobile devices and the challenges this brings. I believe that for mobile networking to work, we need a protocol as simple, general and widespread as XML as the basis for message-passing.

"MOWAHS is a basic research project supported by the Norwegian Research Council in its IKT-2010 program. The project is carried out jointly by the IDI's groups for software engineering (prof. Reidar Conradi, coordinator) and database technology (prof. Mads Nygård). The project will, respectively, have two parts: process support for mobile users using heterogeneous devices (PC, PDA, mobile phones) and support for cooperating transactions/workspaces holding work documents. The project will build upon CAGIS technology and an industrial cooperation is planned." [9]
One key feature of this system is that XML is utilized in the message passing protocol, and since mobile users is to be supported, making sure that even weaker devices are accommodated is a goal.

The motivation for this project is to help extend the range of XML-messages that can effectively be utilized by mobile devices. These devices are much more limited in processor speed, memory capacity, and bandwidth than servers or personal computers. This in turn limits the XML-messages that can be received and interpreted within reasonable time. In addition, for mobile devices, power is a major constraint that should be utilized as optimal as possible. This gives reasons to find energy effective methods of using XML on such devices.

No study as we are aware of has been made on the effect of changing XML-files to reduce parsing-time, communication cost and memory footprint while still keeping the files as XML, so the focus of the project was aimed at finding out which measures can be utilized for these effects. To do this, a statistical analysis was carried out to find parameters in XML-files governing parse time and memory footprint of parsing of XML.

In addition, the following hypothesis stated by Carl-Fredrik Sørensen was tested:
Redundancies in XML can be reduced to optimize the resource usage when using XML in wireless communication and on mobile devices, without compromising the XML properties

A prototype of a system which acts as middleware between the parsers and the application programs used in mobile devices was built. This system reduces the redundancies in the tags of the XML, albeit in a primitive way. In the end, tests of this system were carried out to check the effectiveness of it, giving an indication to the validity of the above-mentioned hypothesis.

The rest of the report is organized as follows:

-Chapter 2: Prestudy: This section will explore the technologies relevant to this project and give a brief description of available technologies.

-Chapter 3: Research Agenda: This section will define the research process used for this project.

-Chapter 4: Own Contribution: This section will outline the work done in this project. The section is divided into two parts: -Finding the parameters.
-Testing the parameters

-Chapter 5: Results: This section will outline the results gathered from doing this project.

-Chapter 6: Evaluation: This section will discuss the result in light of the project goal.

-Chapter 7: Conclusion & Further Work: This section will outline any unfinished business and loose ends the project might have and also give an indication on how this project can be further extended in the future.

Appendixes

-Appendix A: This section will contain all the statistical data from the tests.

-Appendix B: This section will contain some sample XML files used in the testing.

-Appendix C: This section will contain all the source code used throughout the project.

-Appendix D: This section will contain a cd-rom with all the code, both in source and compiled format and all the test results, for easier replication of parts of this project.

7 Prestudy

XML

The markup language known as the Extensible Markup Language (XML) is endorsed by the World Wide Web Consortium (W3C) as a standard for document markup. It defines a generic syntax used to markup data with simple human-readable tags. It provides a standard format for computer documents. [1]

The data is encapsulated in a node-tree in which every node has one and only one parent node. XML can be checked for wellformedness, and only well-formed XML-documents are guaranteed to be treated correctly by programs / parsers. To be well-formed the documents need to adhere to certain rules governing tag-syntax, tag-placements etc. For instance, every tag must begin and end within the same parent tag.

DTD: XML documents may (but does not have to) have a document type declaration (DTD) which declares the legal tags and their behavior to some extent. With a DTD one can determine whether or not the document is valid (adhering to the DTD). XML without DTDs cannot be judged in this way but is still perfectly legal.

Namespace: XML tags may us namespaces which is especially useful when combining several different xml-files into one. Each tag will consist of first a namespace-name, then the tag-name, this means that different ways of treating e.g. a “person” tag is preserved by being able to treat <police_record:person> different from <phonebook:person>

-Schema: The XML Schema language offers facilities for describing the structure and constraining the contents of XML documents, including those which exploit the XML Namespace facility. The schema language, itself represented in XML and uses namespaces, substantially reconstructs and extends the capabilities found in XML document type definitions (DTDs). [10]

-XSL: Extensible Stylesheet Language (XSL) is an XML application that transforms XML documents into a different form that is viewable in web browsers. The language has been split into XSL Transformations and XSL Formatting Objects. The former a general purpose language for transforming one XML document into another for web display, and the latter an XML application that describes the layout of both printed and web pages. [1]

Parsing of XML

In order for any program to make sense of the data contained in an XML message, the message needs to be parsed. To do this the data needs to be passed through a parser which makes the data ready for use, either by a free- or a sequential access scheme.

Today, there are two methods of parsing XML, representing two different paradigms. The first, the Document Object Model, provides a ready data structure for random access, but at a cost. The second, the Simple API for XML, provides cheaper, but only sequential access to the data.

-DOM

The Document Object Model (DOM) method is to parse the entire XML file and make a tree-structure out of the data which can later be traversed and manipulated as one sees fit. This is the original method as proposed by the W3C for parsing XML and numerous implementations exist.

The XML file is first read, then an object tree is generated in the computers memory representing all the nodes in the document and their data. At the root is the document node, followed by the data-tree that the XML represents. Using standard tree-traversal techniques, any node in the object tree can be found.

DOM is considered to be very easy to develop applications for. It is also a standardized and streamlined way of building a data structure out of an XML-file. This gives random access to all parts of the XML-file for as long as is wanted. An XML-file is guaranteed to be a tree with the document node at the root, so this approach to parsing will always work. However, the computer utilizing DOM is required to load the entire XML-file into memory before actions can be taken on it. For some purposes this might be very inefficient.

-SAX

The Simple API for XML was proposed by members of the "DEV_XML" mailing list to solve some of the problems inherent in the Document Object Model way of operating. For the DOM to work, you need to read the entire XML file into memory and keep it there for as long as you need access to parts of it. This applies even if you are only interested in a small part of the XML, e.g. at the beginning. Also, generating the Document Object tree is relatively costly. Especially if you are to make a separate model different from the DOM with the data anyway.

The SAX way of thinking is instead that the XML file is being read sequentially, and events are fired every time an opening tag, closing tag or data is encountered. The actions taken at those points are up to the applications utilizing the SAX-parser, and are specified by the behavior of event-handlers. This way of sequentially firing events means that random access to the XML is not provided. If it is needed, it must be made explicitly by the application. However, many applications do not need random access to the content of an XML file and can do operations on the XML as it is read. Also some applications need to build their own proprietary models of the data, which can just as easily be done from sequential input as from queries in a data-model. The bonus from using SAX in both cases is that you save time and memory by not building the DOM-tree and keeping it in memory for the duration of the parsing.

The SAX parser needs to be set up with an "event handler" it can call methods on as new tags / data are found. The event handler contains the methods detailing what happens when a given tag / given data is encountered and needs to be tailored for each individual application. This way of parsing is also called "push parsing" as the parser actively calls methods (or pushes information) onto the application.

In general, DOM is slow and inefficient but easy to use, SAX is fast and more efficient but hard to use and can easily lead to messy code.

Mobile Devices.

Although new mobile devices are put out to the market every month, they still are considered to be weak devices. To a large extent, this stems from the need to keep power-consumption at a minimum. More memory and faster processors leads to more power-consumption, hence shorter time between each recharge. As battery life is important to users, manufacturers of mobile phones and PDAs equip them with relatively weak processors and little memory compared to personal computers.

This makes mobile devices the weak link when working together with servers or personal computers. This trend is likely to continue, ever widening the gap between mobile devices like PDAs / mobile phones on one hand and personal computers / servers on the other. Despite this drawback, mobile phones with processing capabilities and PDAs are widely used, they are ideal for a wide range of applications where carrying a laptop is either impossible or impractical.

Java2 Micro Edition:

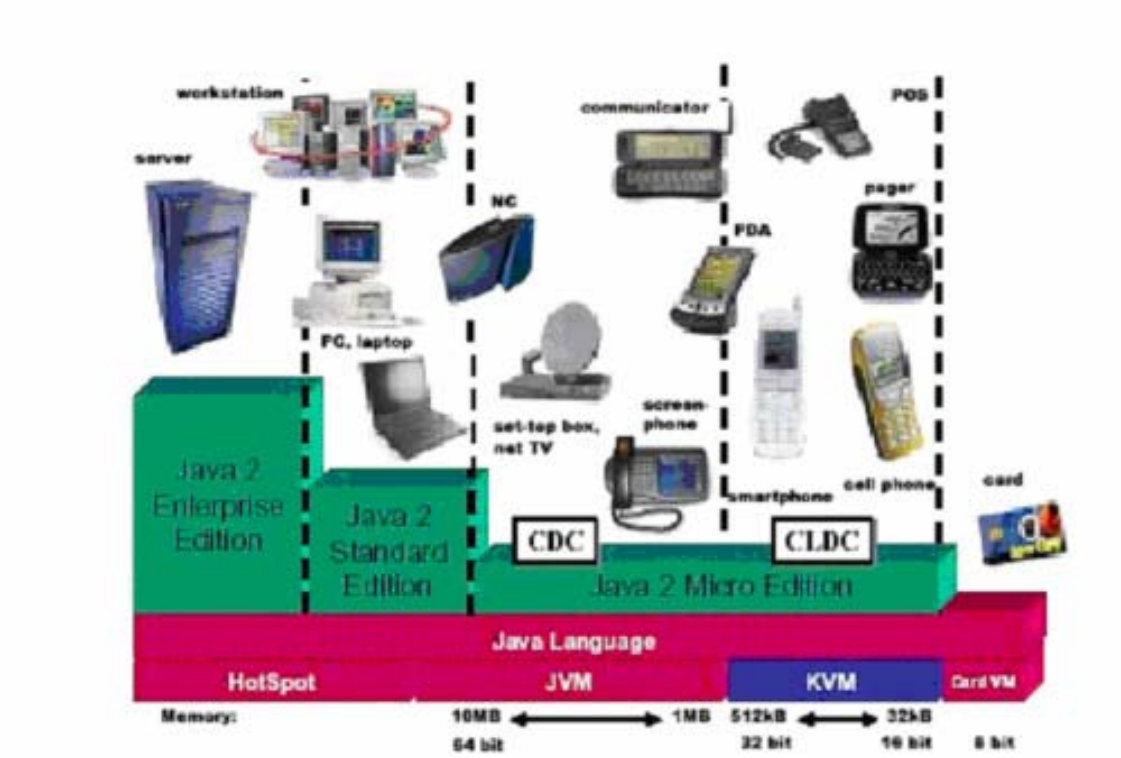

Figure 7.1: Overview of Java technology

The Java 2 Micro Edition (J2ME) is a version of Java specifically designed for use on mobile phones, PDAs and other "weak" devices. Just like the Java 2 Enterprise Edition (J2EE), and Java 2 Standard Edition (J2SE), it requires the code to run on a virtual machine. Unlike J2EE and J2SE, this machine can be either the standard Java Virtual Machine (JVM) or the Kilobyte Virtual Machine (KVM) for the micro edition. The KVM has been designed to leave a small footprint (low memory usage) of between 70K and 250K. This makes it possible for smaller devices to utilize Java without being overpowered.

J2ME is in addition organized into the virtual machine, configuration and profile layers as shown in figure below. [The configuration] "…defines the minimum set of Java virtual machine features and Java class libraries available on a particular "category" of devices representing a particular "horizontal" market segment." [8]
The configurations are granular enough that they encompass large chunks of the market, offering a core of functionality that will be the same on any of the various devices within a chunk.

For the time being, only two configurations exist, the Connected Device Configuration (CDC), offering support for more powerful PDAs and hardware that are constantly connected to network and servicing users. And the Connected Limited Device Configuration (CLDC), which is a common denominator for all mobile phones and weaker PDAs supporting the Java language.

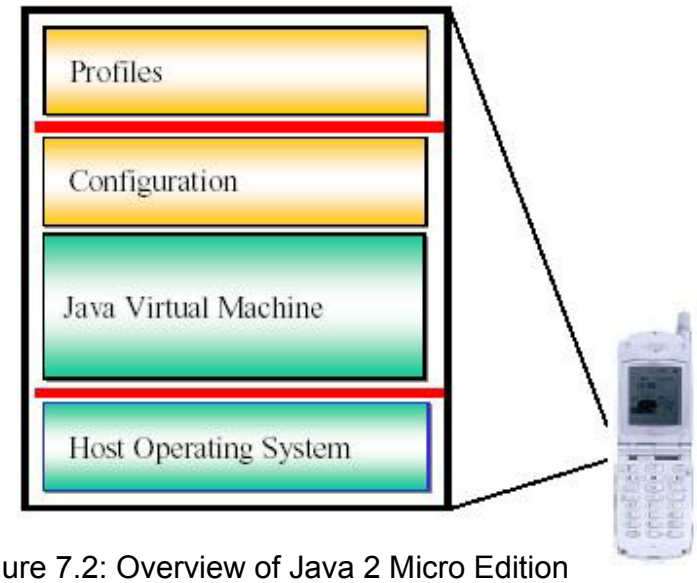

Figure 7.2: Overview of Java 2 Micro Edition

On top of the Configurations we find the Profiles, these are extensions of the configurations attuned to the specific hardware of any given device. This is what actually defines the Java class libraries and methods available on a specific device series, for example the Nokia phones.

Since more and more mobile phones and PDAs support some configuration of J2ME (usually CLDC), it is most likely that a system for extending usable size in XML messages will need to be written in Java. Since such a system particularly needs to be compatible with weak devices it will need to be designed using the CLDC configuration in J2ME.

Redundancy in XML:

XML files contain potentially many repetitions of long tag-names with equally long end-tags. For instance a XML-ised version of the yellow pages would potentially contain millions of the <ADDRESS> … </ADDRESS> tag. This can lead to
Files that are much larger than the data they contain. This redundancy can be reduced if, say, instead of using the <ADDRESS>…</ADDRESS> pair, an <A>...</A> pair is used instead. The resulting XML will still be equally valid XML, but the tags will no longer be as human-readable. For some applications this can be an acceptable way to save space. In order for the XML to be usable, however, some schema must be devised to translate the XML back to its original form before the application program reads the data.

Reducing redundancy in XML:

To reduce redundancies in XML, a system needs to be in place which translates back and forth between the original XML and the reduced XML. This is important since the application programs should not need to be aware that such an optimization has taken place.

Such a system will need to be put between the server and the client such that the server (strong device) can output XML which then are reduced before they are sent to the client (weak device). The best way to implement this is as a middleware component, so that the server programs are unawares any optimization has happened. Otherwise different versions of the server programs must be written for different types of clients.

A conceptual model of such a system would be like figure 7.3:

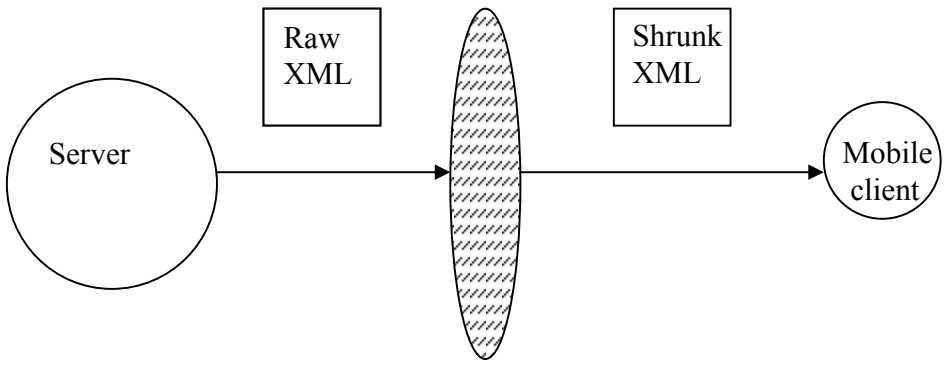

Figure 7.3: Conceptual model of tag reducing system, server side

Possible ways to execute redundancy-reduction:

Different parts of the XML can have redundancies which could be optimized; the tag names, the attribute names, the attribute values and the data. As such, four different schemes for optimizing XML can be suggested, combinations are also possible:

1: reduce tag names to single letters
2: reduce attribute names to single letters
3: reduce common attribute values to single letters
4: reduce common data values to single letters

Consequences of different ways of redundancy-reduction:

If these schemes are used to reduce various redundancies, different requirements are put on the XML. Here is a list of what is needed for the different schemes:

1: must have a scheme embedded in the XML to get back original tag name
2: must have a scheme embedded in the XML to get back original attribute name
3: must have a scheme embedded in the XML to get back original attribute value
4: must have a scheme embedded in the XML to get back original data value

A possible scheme addressing all of 1, 2, 3 & 4 would be to include in the XML at the root level a header tag in which all translations are included:
For example, the XML shown in figure 7.4:

```
<?xml version=”1.0” ?>

<header>
        <person>p</person>
        <name>n<name>
        <phone number>h</phone_number>
        <address>a</address>
        <sex>s</sex>
        <male>m</male>
        .
        .
        .
</header>
<p s=”m”>
        <n>Ola Nordmann</n>
        <a>Trondheimsveien 12 b,
        7000 Trondheim, NORWAY</a>
        <h>01234567</h>
</p>
.
.
.
```

Figure 7.4: Example of tag reduced XML

Would be a way of saying that each time a “p” is encountered in a tag it should be translated into “person” etc.
The same principle would apply to both attribute names, attribute values and data.

To use this system at the client side, it needs to be placed between the parser and the application program. This way, neither the parser nor the application program need be aware of the reduction scheme.
For SAX parsing, whenever the parser sends an event, that event can be translated in the system before being sent to the application program.
For DOM parsing, whenever the application makes a request to the DOM-structure, that request can be translated before reaching the parser.

The architecture will need to be like figure 7.5:

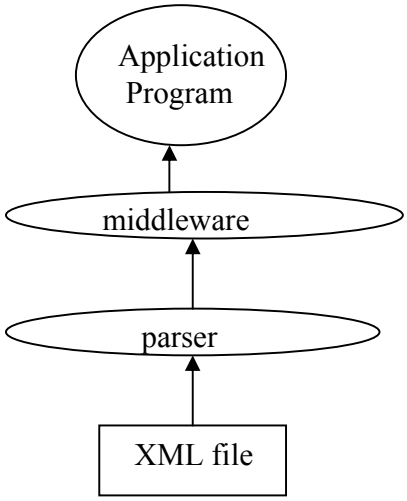

Figure 7.5: Conceptual model of tag reducing system, client side

This can be implemented rather easily in an event handler for SAX which translates the events and then throws them further up to the "real" event handler utilized by the application program.

It can also be implemented rather easily in a wrapper class for the DOM parser.

In practice, the application program initializes the middleware which in turn initializes the "real" parser. The real parser would think that the middleware is the application program, and the application program would think that the middleware is the parser.

8 Research agenda

This section details the research plan of this project

This project is concerned with identifying parameters that make XML harder/easier to parse, and also to test whether tags can be reduced to make the parsing easier. To do this, the project has in part been structured as a software experiment. [4]

To make this project repeatable on mobile devices, everything is done in Java in the J2SE edition. This means that all the code written will be almost directly transferable to mobile devices, using J2ME instead of J2SE at a later stage.

As mentioned earlier, mobile devices have limited memory, power and bandwidth. For the testing of these, This project will concentrate on the two former, as bandwidth is directly dependent on the raw size of an XML-message only, and the communication cost can easily be calculated given a message size.

To measure power usage, the processing time needed to parse a given XML file will be used as measurement. This is done because the longer a file takes to be parsed, the more power is needed in the process.

To measure memory usage, Java's own memory measurements will be used. Although coarse in granularity, it will nevertheless give important answers regarding correlations.

For the testing part, the project has been structured as a software experiment as detailed by Wohlin et al. [4]

According to Wohlin et al, the phases typically involved in software experiments are:

- Experiment definition:
- Experiment planning:
- Experiment operation
- Analysis and interpretation
- Presentation and package
- Conclusions

In the definition phase, one typically state what the purpose of the study is, and define the goal of the study. A null hypothesis and alternative hypotheses are formulated in this phase.

In the planning phase, the context of the experiment is determined in detail. This includes setting up testing facilities and determining what is to be tested and how.

In the operations phase, the experiment environment is set up according to what has been determined in the planning phase. Actual experiments are run and data collected.

In the analysis and interpretation phase, descriptive statistics are used to understand the data informally. More detailed statistical analysis is done on data deemed to be relevant. The null hypothesis is either rejected or accepted.

In the presentation and package phase, the results are written down and made available to the public (this report). Also preserving a lab package for easy replication of the experiment is common. The lab package is to be found in appendix D.

In the conclusions phase, the validity of the results gained is discussed and consequences derived from the results are identified.

All of these phases were used explicitly, with the exception of the presentation and package phase which can be seen as report.

Each of these phases will be further outlined in the Own Contributions section of this document with the exception of the conclusions phase, which will be located in the Results section of this document..

9 Own Contribution

9.1 Finding Parameters to test:

According to Wohlin et al. “We can only test what we already know/suspect.” [4]
So finding candidates for parameters in XML that regulate parsing cost is essential.

9.1.1 Literature study:

After searching throughout available literature, the only conclusion available was that this is a relatively new problem that has not been made available though research papers or textbooks. Neither in textbooks, research papers, nor the Internet was any information on what factors make XML resource-intensive to parse. Numerous sources compare parsers to each other, but always without giving thought to how the test-data they parse affect the parsing times.

9.1.2 Educated guesses

Since the literature study provided no clues as to what determines the cost of parsing XML, educated guesses towards factors are necessary to make.

Three possible factors seem like logical choices. These can be divided into two groups, namely size and complexity. Size is an obvious choice, since the more information is present, the more time and memory one would assume was spent parsing it. In addition, there might be some factors that govern how complex a document is to parse.

-Raw size in bytes:
Since each byte needs to be read at least once, it makes sense that the more bytes are present, the more time will be used to read them. Also, the more bytes are present in an XML file, the more memory is needed to “juggle” with them.

-Number of tags:
For SAX-parsers, each time a tag is encountered a callback-method is fired, so one should assume that this has an impact. For DOM parsers, each tag is made into its own node in the document tree, so this also should have an impact.

-Average level of bytes:
This is a suggestion on measuring complexity. The idea is that the further down into the XML-tree a node is, the more expensive it is to manipulate. Using just the maximal depth of a node might be too sloppy since parsing of documents that have a deep maximum level, but where most of the information is located almost at top level would most likely be to a large degree independent of the max depth. Instead of using the maximum level, I therefore use the average level.

In addition to these three factors, a refinement of the raw size hypothesis, reduction of tag redundancies was tested.

XML is almost bound to have a lot of wasted space in the tag-names, which are repeated for every element. If these tag-names are changed to something very short, there is a huge potential for reducing the size of messages.

As I have found no indications through my literature search on which factors might affect processing speed and memory footprint, I will let that be my null hypothesis. "*The processing time and footprint of XML-parsing is constant."*

The alternative hypothesis will be the other possible factors I have identified.

9.2 Experiment

9.2.1 Experiment definition:

The Goal Question Metric model is a standard way of defining experiments. By using GQM the project is focused towards a goal and it becomes easier to verify that important aspects of the experiment are defined before the planning and execution take place.

Using the Goal Question Metric models template, the definition of the experiment is:

Object	XML-parsers and XML-files
Purpose	find parameters that affect the time and memory usage for parsing XML-files and test whether redundancy reduction reduces time and memory usage.
Quality focus	time and memory usage in parsing of XML-files
Perspective	Irrelevant.
Context	Laptop-computer. If time: on PDAs / mobile phones

Table 9.2-1: Project definition.

The project has three parts, find the parameters for time usage, find the parameters for memory usage and test whether there is any point in doing redundancy reduction.

9.2.2 Experiment planning

Based on the educated guesses made earlier, the null hypotheses and some alternative hypotheses can be formalized:

Null Hypothesis 1: All XML files are parsed with equal cost in terms of time used.

Null Hypothesis 2: All XML files are parsed with equal cost in terms of memory used.

Alternative hypotheses:

1-Time usage:	1-A	The raw size of the XML-file determines the time used to parse it.
	1-B	The number of tags in the XML-file determines the time used to parse it.
	1-C	The average tag level in the XML-file determines the time used to parse it.
	1-D	Tag redundancy can be utilized to make parsing of XML cheaper in terms of time used.
2-Memory needed:	2-A	The raw size of the XML-file determines the memory needed to parse it.
	2-B	The number of tags in the XML-file determines the memory needed to parse it.
	2-C	The average tag level in the XML-file determines the memory needed to parse it.
	2-D	Tag redundancy can be utilized to make parsing of XML cheaper in terms of memory needed.

Table 9.2-2: Alternative hypotheses.

Testing Environment:

The environment in which the tests will be run is a laptop computer of type Targa Extender 400 with 256 MB RAM, running Windows XP.
The reason for this choice of environment is that this leaves greater confidence that all system variables will remain relatively constant and affect the experiment little or not at all. Alternatively, the tests could have been run on much larger and faster computers available at the University of Science and Technology, Trondheim, but this would give little control of the amount of measurement error input into the system by other users.

For these tests, Java 2 Standard Edition (J2SE) will be used. The reason for this (as has been explained earlier) is that a later transition to Java 2 Micro Edition (J2ME) is to be possible at a later stage. When transferring the code to J2ME, only minor changes needs to be made, and the parsers and testing framework will work as it does now under J2SE.

Parsers used during testing of hypothesis A-C will be:

-Xerces (SAX): The SAX/DOM parser for Java developed by the Apache community in The Apache XML Project is one of the most complete parsers freely available for Java. The parser conforms to the Java API for XML Processing (JAXP) defined by Sun Microsystems. [13]

-Piccolo: The Piccolo Parser is a lightweight SAX parser that claims to work "extremely fast" and which also conforms to the JAXP standard. Piccolo is open source software under the Gnu Lesser General Public Licence. [12]

-kXML: kXML is a small XML pull parser, especially designed for constrained environments such as Applets, Personal Java or MIDP devices. [15]
kXML is based on the XML pull API [16].

Parsers used during testing of hypothesis D:

-Xerces: SAX-parser (see above)

-Piccolo: SAX-parser (see above)

-Xerces: DOM-parser. (see above)

The reason the kXML parser is not used for testing hypothesis D is that to do so the translation module would have to be implemented very differently from the two other SAX parsers which follow the JAXP – standard. This might have lead to discrepancies which could have undermined any results from the tests.

The reason I use the Xerces DOM parser for testing hypothesis D is that it is interesting to see what effect smaller tags will have on the memory usage in a Document Object Model tree It would also be interesting to see how it would measure up in testing hypothesis A-C, but this was not done.

VARIABLES:

Testing alternative hypotheses A, B and C is different from testing alternative hypothesis D. In the first three hypotheses, all factors except one will be kept constant and tests will be run while varying that one factor. Due to limited time, a thousand tests on 3-5 different values for each of the factors in question will be run.

The variables to be considered are:

-raw size of XML-file
-number of tags
-average level of bytes.

First random XML-data that are of the size/structure needed for each of the tests is generated, then the test is run. For each class of data this will be done 1000 times to create enough statistical data to work with.

In testing hypothesis D, XML with a lot of tag redundancy will be created of 5 different sizes, then that data will be tag-reduced. Tests will be run on both the original and the reduced XML data and measures will be made on how the reduced files compare to the original ones in regards to parsing time and parsing memory footprint.

SUBJECTS

Testing hypothesis A, B and C:

As stated earlier, the data has been made so that each parameter; size, number of tags, and average tag-level can be assessed independently. To do this, the XML-files are generated in batches in which one parameter is varied whilst the other two are kept constant.

Testing hypothesis A:
In testing alternative hypothesis A, two different batches are run:

1)
Number of tags: 1
Average tag-level: 1
Size: {0.1kB. 1kB, 10kB, 100kB}

2)
Number of tags: 100
Average tag-level: 4
Size: {1kB. 10kB, 100kB, 1Mb}

Testing hypothesis B:
In testing alternative hypothesis B, two different batches are run:

1)
Number of tags: {1, 10, 100}
Average tag-level: 1
Size: 1kB

2)
Number of tags: {10, 100, 1000, 10000}
Average tag-level: 4
Size: 100kB

Testing hypothesis C:
In testing alternative hypothesis C, two different batches are run:

1)
Number of tags: 100
Average tag-level: {2,4,8,16}
Size: 1k

2)
Number of tags: 10000
Average tag-level: {4, 8, 16, 32, 64, 128}
Size: 100kB

With these constellations, the effect of the three factors should be testable independently from each other. Other constellations could be imagined, but these will provide enough data to work with. Any other present factor in the XML files will hopefully be eliminated as the tests are done on random data that should have nothing more in common than being of the same data-class in respect to size, average tag level and number of tags.

Testing hypothesis D:
In testing hypothesis D, data of a different form is needed. The data must contain much redundancy for it to be used in testing redundancy reduction. For this purpose, a different XML generator was made. The XML outputted will be of a form one would expect in a XML - ised yellow pages data file. 5 different sizes will be tested: 1, 10, 100, 1000 and 10’000 posts of data. The XML will first be parsed, then reduced, then parsed again with a reduction scheme

The output measured will be both the maximum memory usage during the parsing, since this greatly affects which devices can and cannot parse a given file, and the time it takes to parse.

This leads to 25 different data-inputs X 1000 tests each X3 different parsers = 75’000 different test runs for hypothesis A-C and 10 different data inputs X 1000 tests each X 3 parsers = 30’000 different test runs for hypothesis D.

In the end 105’000 tests will be done, each giving information on time and max memory used. This should be sufficient statistical data to do analysis on.

9.2.3 Experiment Operation:

The testing tools used have been developed as part of the project.

-The GenerateXML-program generates random well formed XML-data that conforms to the data-specifications as given in the experiment definition. In this way the data is guaranteed to not be biased in any way.

-The ParseTimer runs one parsing of one file using one parser. It also outputs the physical time used to do so to a file along with the peak value of the memory footprint

-The different parsers parse the XML file. These are called Parse*

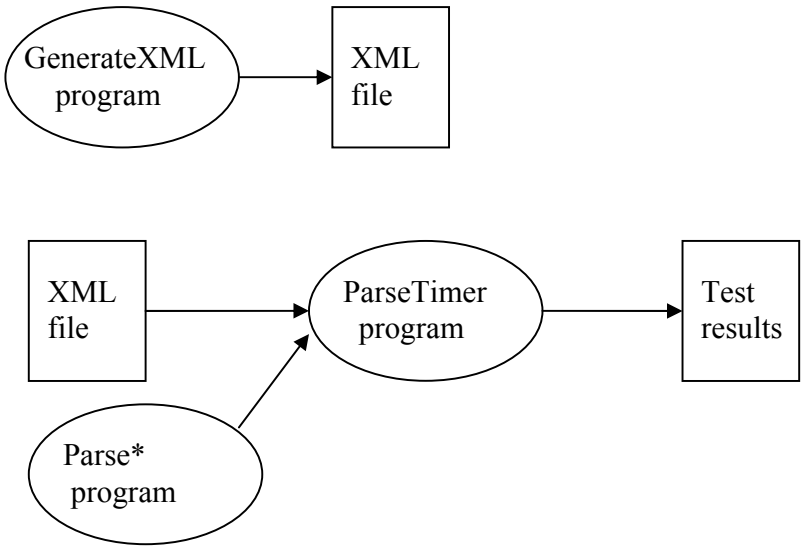

Figure 9.2-1: Overview of test system

The test results can then be analyzed using statistical analysis.

9.2.4 Experiment Analysis & Interpretation

First the data collected will be examined informally to see if some trends can be gathered. Then statistical methods will be applied to see if any trends can be proven. To do this, the T-test will be used. Throughout this section, the acronyms ksax, xsax, picc and xdom are used for the kXML parser, the Xerces SAX parser, the Piccolo parser and the Xerces DOM parser respectively.

The complete statistical data from the tests can be found in Appendix A.

Testing hypothesis A:

Figure 9.2-2 and 9.2-3 both show the mean values found from doing 1000 tests with varying the size of xml-files while keeping other factors constant.
-In test run A, the number of tags was 1, and the average tag level was 1
-In test run B, the number of tags was 100, and their average tag level was 4

-Effect on parsing time:

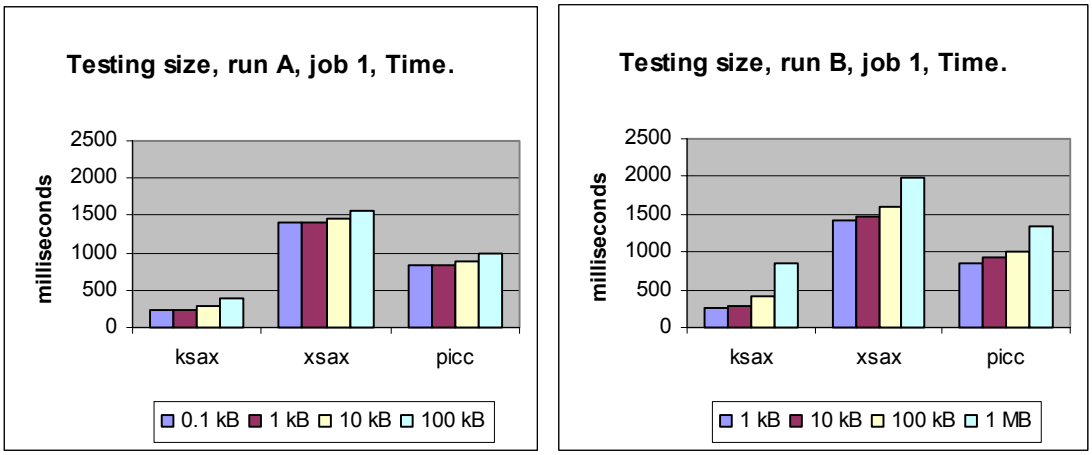

Figure 9.2-2: Results from testing hypothesis 1A

As can be seen from the graphs, there is only a slight increase in the processing time for the test run A from the smallest to the largest files. For test run B there is a noticeable jump from the 100kB file to the 1MB file. This could mean that XML-files below 100k are all parsed at the same speed, i.e. that doing the parsing is much quicker than the overhead of setting up the parsing.
Thus we can say that the size of XML files only affects the parsing time once the file gets big enough. It would seem that this threshold lies somewhere between 100kB and 1MB

-Effect on memory usage:

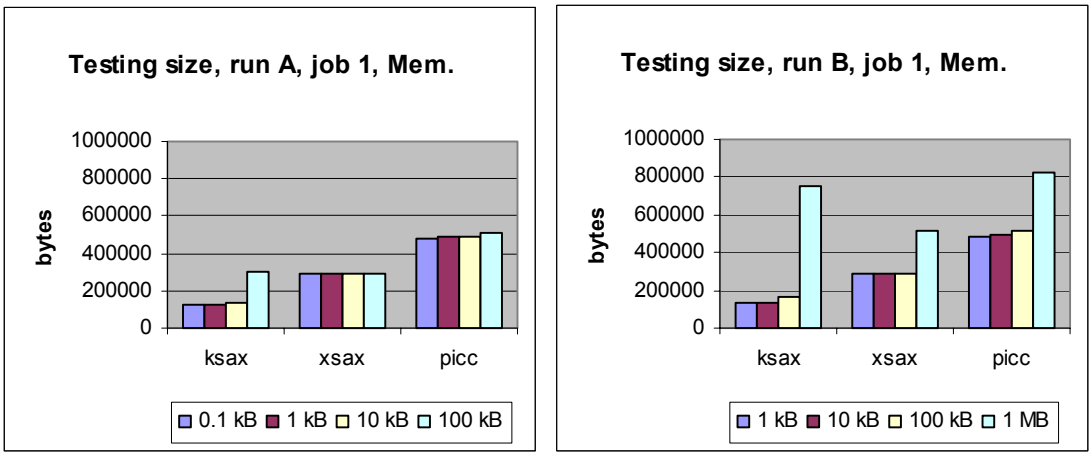

Figure 9.2-3: Results from testing hypothesis 2A

As can be seen from the graphs, the two regular SAX push-parsers have a fairly constant memory usage independent from the size of the XML-file for all of test run A, and for most of test run B. Only when the size of the XML-file reaches 1 MB do these two parsers require more memory. The kXML pull-parser on the other hand behaves a little erratic, for test run B it behaves like the other two, making a major jump upon parsing the 1 MB file, but it does a similar jump in the test run A, although not as large.
On one hand it manages to parse a 100k file with about 200k memory used (run B), but on the other it needs 300k memory to do the same job with a different file of similar size. Why this happens would be interesting to investigate.
It might mean that the pull parser is dependant on other aspects than just size, or that the combination of a 100kB file with only one tag is especially bad. In either case it should be investigated further in a different project why this strange result was made.

All of the parsers agree though, that parsing the 1MB file is a much larger job than the other ones. Hence we can say that similar to time usage, memory usage is unaffected by XML file size below a certain threshold between 100kB and 1MB.

Testing Hypothesis B:

Figure 9.2-4 and 9.2-5 both show the mean values found from doing 1000 tests with varying the number of tags in xml-files while keeping other factors constant.
-In test run A, the size of the XML file was 1kB, and the average tag level was 4
-In test run B, the size of the XML file was 100kB, and the average tag level was 16

-Effect on parse time:

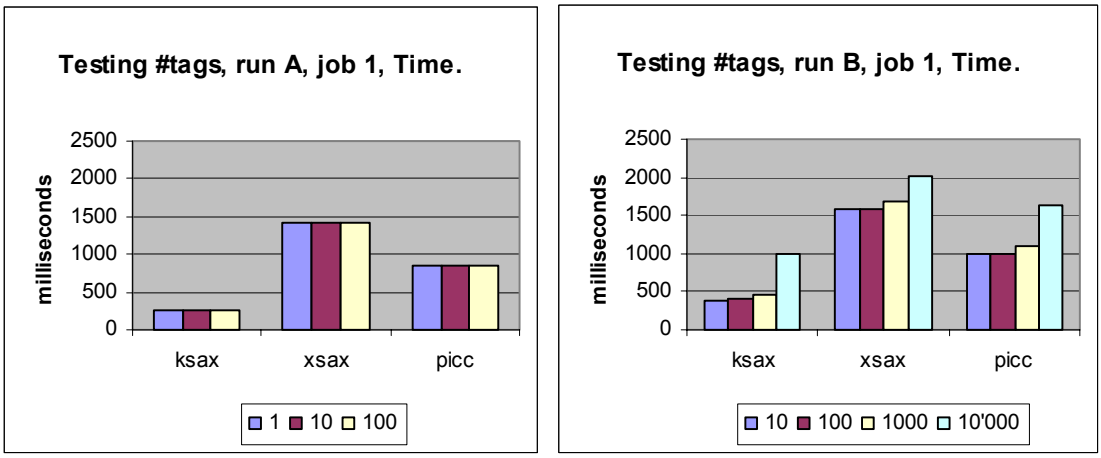

Figure 9.2-4: Results from testing hypothesis 1B

As with size, there does not seem to be any effect on parsing time under a certain threshold. All the parsers agree that when the number of tags stay below 1000, the time to parse is constant, however, when parsing 10’000 tags in test run B, all the parsers does a noticeable jump. This can again signify that below a certain amount, the number of tags does not affect the time needed to parse the XML file, but above a certain threshold, which lies somewhere between 1000 and 10’000 tags this starts to have an effect.

-Effect on memory:

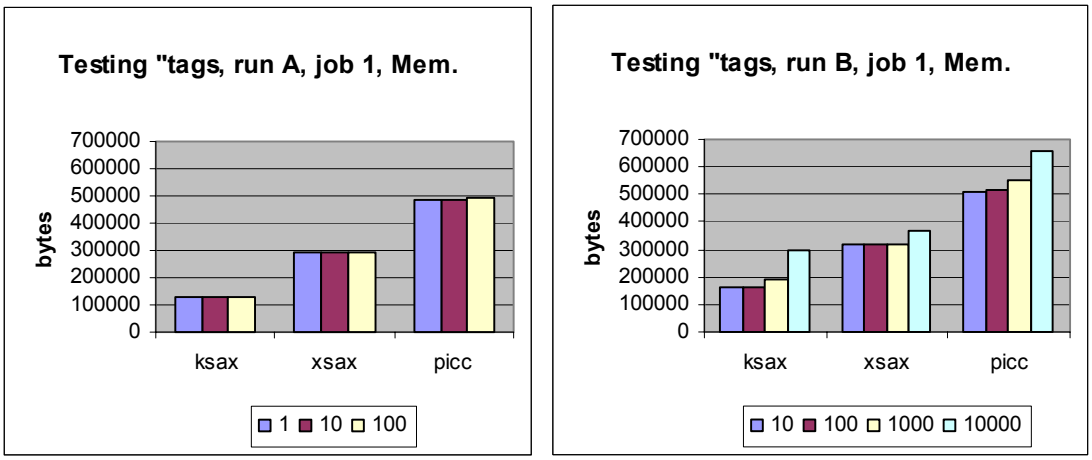

Figure 9.2-5: Results from testing hypothesis 2B

As for the parsing time, there does not seem to be any effect on the memory needed for XML messages with number of tags under a certain threshold. All the parsers agree that when the number of tags stay below 1000, the time to parse is constant, however, when parsing 10’000 tags in test run B, all the parsers does a noticeable jump, the Piccolo and kXML parser especially. This can again signify that below a certain amount, the number of tags does not affect the memory needed to parse the XML file, but above a certain threshold, which lies somewhere between 1000 and 10’000 tags, the number of tags starts to have an effect.

Testing hypothesis C:

Figure 9.2-6 and 9.2-6 both show the mean values found from doing 1000 tests with varying the average tag level of xml-files while keeping other factors constant.
-In test run A, the size of the XML file was 1kB, and the number of tags was 100
-In test run B, the size of the XML file was 100kB, and the number of tags was 10k

-Effect on parse time:

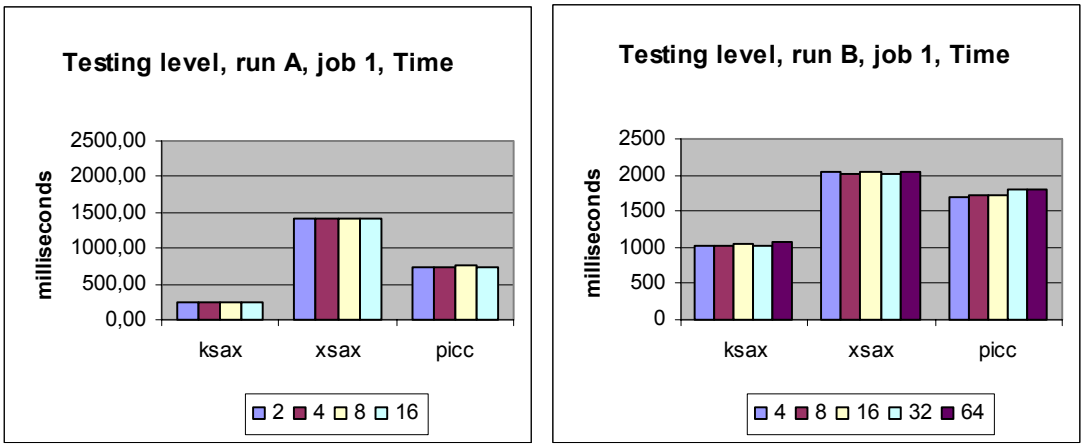

Figure 9.2-6: Results from testing hypothesis 1C

As can be seen from these graphs, the average level of tags does not seem to have any influence on the time needed to parse the XML files. The slight variations in the samples most likely stems from other sources like measuring error, than any real correlation between average tag level and time needed to parse XML files.

-Effect on memory usage:

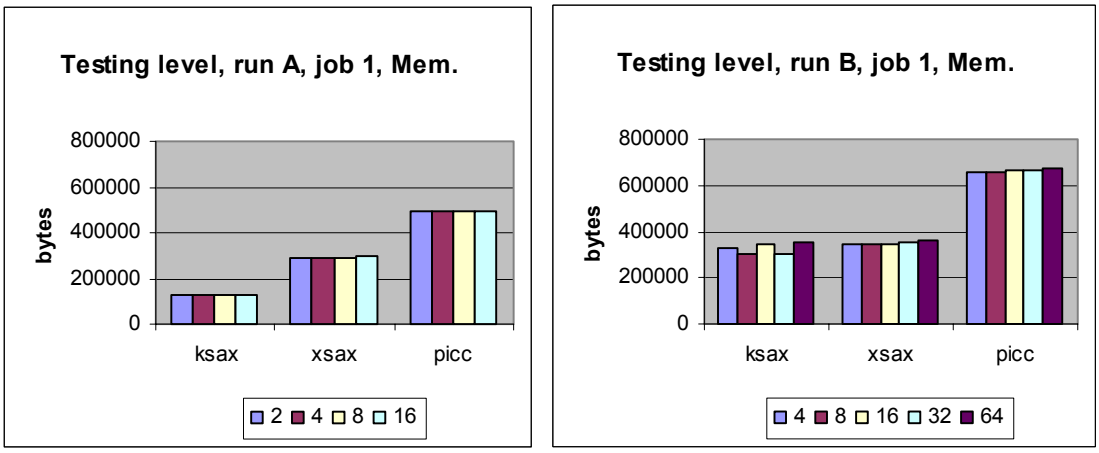

Figure 9.2-7: Results from testing hypothesis 2C

As can be seen, the level of tags in an XML file does not seem to have any effect on how much memory is needed to parse the files. The slight variations in the samples most likely stems from other sources like measuring error, than from any real correlation between average tag level and amount of memory needed to parse XML files.

Testing hypothesis D:

Figure 9.2-8 and 9.2-9 both show the mean values found from doing 1000 tests of five different XML files, then doing the same tests with files in which the tag redundancy had been reduced.

-Effect on parse time:

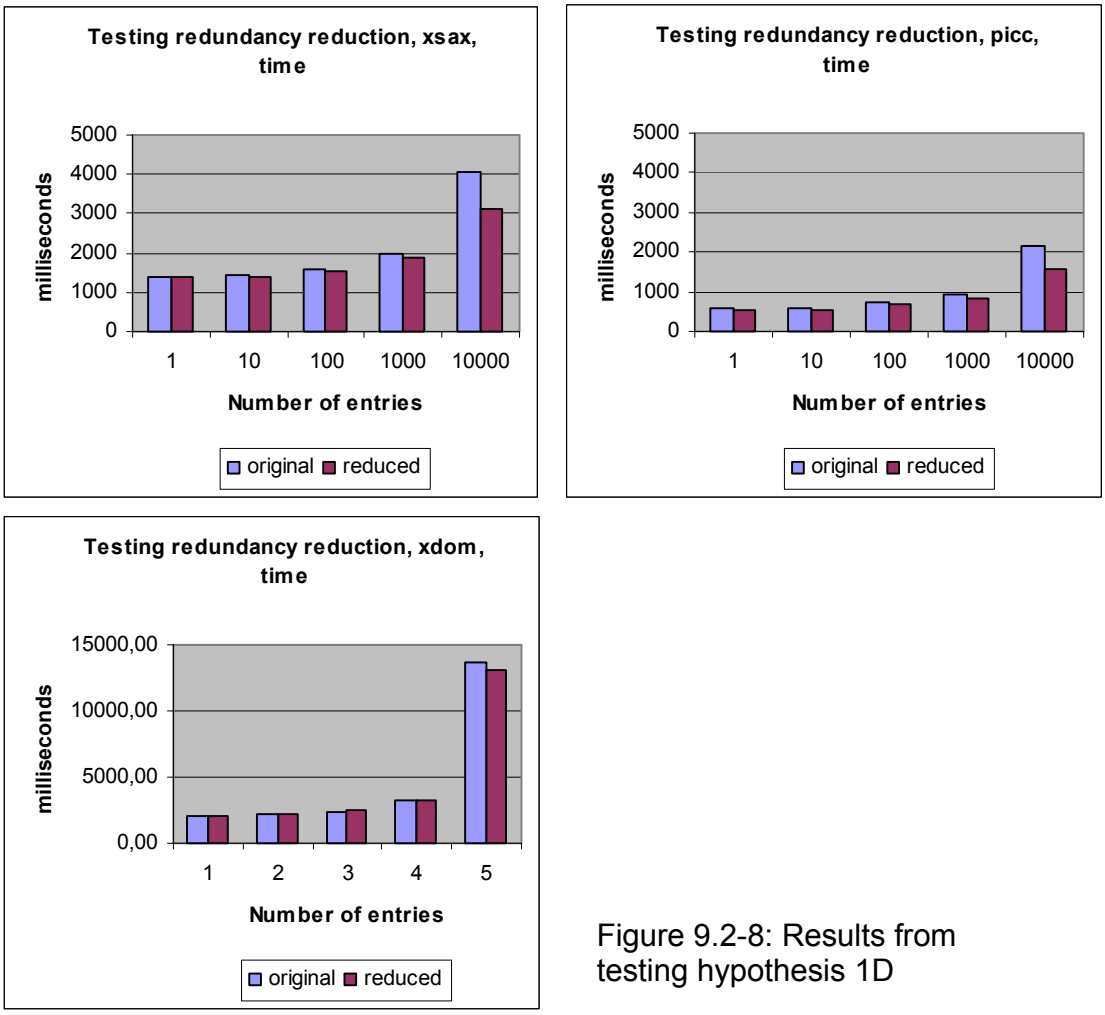

Figure 9.2-8: Results from testing hypothesis 1D

As can be seen in the graphs, for the SAX parsers there seem to be a slight gain in parse time when the redundancies have been reduced, this gain seem to be increasing with the size of the XML messages. For the DOM parser, some gains are made, but only when the size of the XML message is large enough.

-Effect on memory needed.

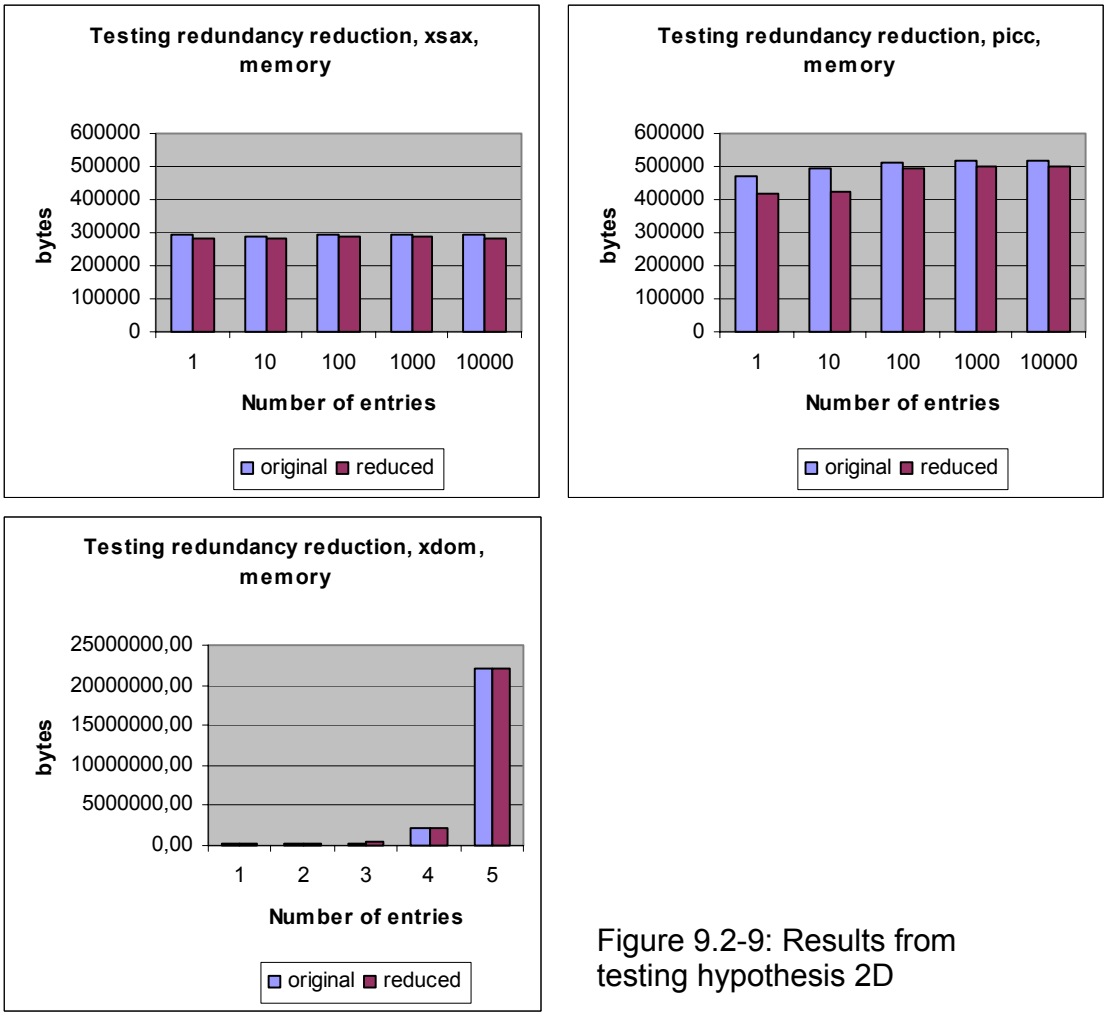

Figure 9.2-9: Results from testing hypothesis 2D

The Xerces SAX parser has a very stable memory usage almost independent from the size of the XML. The Piccolo SAX parser needs consistently less memory for the reduced files than their original counterparts. The Xerces DOM parser actually uses more memory to treat the reduced files when they get large.

-Effect on communication cost:

In this redundancy-reduction scheme, it is also interesting to see the effects on communication costs. Since we are shrinking the size of the XML, the cost of transmitting this XML file is made cheaper. This is because the communication cost of transmitting a file over either a cable or a wireless medium is directly proportional to the size of the file.

File	Original size	Reduced size	Percent wise reduction
1 post	465	487	-4,7%
10 posts	4950	2514	49,2%
100 posts	51360	23552	54,1%
1000 posts	512360	235261	54,1%
10000 posts	5180656	2395368	53,8%

Table 9.2-3: Effect on comm. cost when reducing tags.

As can be seen in table 9.2-3, for this type of file (yellow pages mockup), because it contains proportionally a lot of redundant tags as opposed to data, the average reduction in size is above 50% for all the files. The only noteworthy exception is the first file which, because it only contained few tags, even though these were reduced in size, the total size of the file increased. This happened because the translating information was included in the file, and this increased the size enough to override any gain gotten from smaller tag-size. For the other files, the gain was much larger, so that the relative low constant cost of adding the translation information could but be ignored.
It would be interesting to find out exactly how much is to be gained from reducing tags by a mathematical formula based in parameters in XML.

9.2.5 Analysis and interpretation:

As seen in the informal analysis, there does not seem to be any correlation between the average tag level and parse time or memory usage. There does, however seem to be some truth to alternative hypothesis A and B, in that when the size gets big enough and the number of tags big enough, they seem to affect the parse time and memory needed to parse an XML file. These claims will now be tested more thoroughly using the T-test:

To test the hypotheses, for all three parsers a statistical test will be made to see whether or not the two extremes in test set B have significantly different parse times and memory used.

When doing the T-test, the result is a decision whether or not the largest value is significantly larger than the smallest.

Using the terminology: -The smallest of the test cases is called X
-The largest of the test cases is called Y
-Number of tests on test case X is called n
-Number of tests on test case Y is called m
-Variance in tests on test case X is called: S^2_x
-Variance in tests on test case Y is called: S^2_y
-Mean value in tests on test case X is called: x'
-Mean value in tests on test case Y is called: y'

we first need to find $S_p = \sqrt{ ((n-1)S^2_x + (m-1) S^2_x) } / \sqrt{ (n + m - 2)}$

then calculate $t_0 = (x' – y') / S_p \sqrt{(1/n + 1/m)}$

when doing more than 120 tests (In this project 1000 tests are run), the t_0 value is evaluated in the following way: If t_0 is greater than 1.960 then the measured difference between the two sets are significant, and one can say that the two sets are really different with a margin of error of 5%. If t_0 is less than 1.960 then there is no statistical proof to say that the two sets are different. The former case results in throwing away the null hypothesis, the latter results in accepting it. [5]

Testing hypothesis 1A: *The raw size of the XML-file determines the time used to parse it*.

t_0 values:

kXML SAX:	197,7034
Xerces SAX:	84,29967
Piccolo SAX:	68,25542

Table 9.2-4: T-test values for hypothesis 1A

The results from the analysis is that for all parsers, it takes significantly more time to parse larger XML files than smaller ones. This means that the null hypothesis must be rejected and alternative hypothesis 1A must be accepted. In other words, we can say that the size of XML messages do have an impact on the time needed to parse them.

Testing hypothesis 1B: *The number of tags in the XML-file determines the time used to parse it.*

t_0 values:

kXML SAX:	190,7845
Xerces SAX:	63,41396
Piccolo SAX:	66,04868

Table 9.2-5: T-test values for hypothesis 1B

The results from the analysis is that for all parsers, it takes significantly more time to parse XML files with many tags than ones with few tags. This means that the null hypothesis must be rejected and alternative hypothesis 1B must be accepted. In other words, we can say that the number of tags in XML messages do have an impact on the time needed to parse them.

Testing hypothesis 2A: *The raw size of the XML-file determines the memory needed to parse it*.

t_0 values:

kXML SAX:	239,7491
Xerces SAX:	64,19935
Piccolo SAX:	80,12312

Table 9.2-6: T-test values for hypothesis 2A

The result from the analysis is that for all parsers, it takes significantly more memory to parse larger XML files than smaller ones. This means that the null hypothesis must be rejected and alternative hypothesis 2A must be accepted. In other words, we can say that the size of XML messages do have an impact on the memory used in parsing them.

Testing hypothesis 2B: *The number of tags in the XML-file determines the memory needed to parse it*.

t_0 values:

kXML SAX:	85,4481
Xerces SAX:	22,4378
Piccolo SAX:	71,70803

Table 9.2-7: T-test values for hypothesis 2B

The result from the analysis is that for all parsers, it takes significantly more memory to parse XML files with many tags than ones with few tags. This means that the null hypothesis must be rejected and alternative hypothesis 2B must be accepted. In other words, we can say that the number of tags in XML messages do have an impact on the memory used in parsing them.

Testing hypothesis 1D: *Tag redundancy can be utilized to make parsing of XML cheaper in terms of time used.*

Testing hypothesis D is slightly different from testing A-C in that the possible gains in parse time and memory is to be measured pairwise between the original and the reduced files.

Table 9.2-8 contains the t_0 values derived from testing whether parsing the original file takes longer time than parsing the reduced file. Again values of above 1,960 means that we can (with 5 percent chance of error) say that this is true. The larger the t_0 value the more likely it is that parsing a reduced file is cheaper than parsing the original file.

Parser:	File	t_0-value
Xerces SAX	1 post	3,03
	10 posts	7,42
	100 posts	11,19
	1000 posts	14,49
	10000 posts	99,44
Piccolo SAX	1 post	11,17
	10 posts	15,63
	100 posts	16,32
	1000 posts	23,16
	10000 posts	88,10
Xerces DOM	1 post	-4,15
	10 posts	-3,89
	100 posts	-1,40
	1000 posts	2,08
	10000 posts	14,37

Table 9.2-8: T-test values for hypothesis 1D

From table 9.2-8 we can see that the two SAX parsers consistently parse the files faster when they are redundancy-reduced. This most probably stems from the fact that the parsers reads through the file sequentially, and when everything has been read they are done. The larger a file is, the longer it will take for it to be physically read from the hard drive. We also see that as the files get larger, there is a more definite gain in terms of time used when reducing them first.

However, for the DOM parser, parsing a reduced file might actually be more expensive in terms of time used than parsing its original counterpart. Only when the file gets big enough (the size of the reduced 1000 post file is 235’261 bytes) is any gain noticeable. This could stem from the overhead of maintaining the translation module in memory and doing translations on everything done.

Testing hypothesis 2D: *Tag redundancy can be utilized to make parsing of XML cheaper in terms of memory needed.*

Table 9.2-9 contains the t_0 values derived from testing whether parsing the original file takes more memory than parsing the reduced file. Again values of above 1,960 means that we can (with 5 percent chance of error) say that this is true. The larger the t_0 value the more likely it is that parsing a reduced file is cheaper than parsing the original file.

Parser:	File	t_0-value
Xerces SAX	1 post	3,16
	10 posts	3,12
	100 posts	1,32
	1000 posts	1,29
	10000 posts	5,28
Piccolo SAX	1 post	11,70
	10 posts	16,42
	100 posts	8,50
	1000 posts	9,03
	10000 posts	8,49
Xerces DOM	1 post	-1,85
	10 posts	0,12
	100 posts	-10,88
	1000 posts	-6,44
	10000 posts	-4,55

Table 9.2-9: T-test values for hypothesis 2D

Whereas the Piccolo SAX parser consistently gives better results on the reduced file, the Xerces SAX parser only gives significantly better results on the smallest and largest files. This could mean that whether or not reducing the redundancies has any effect in SAX parsing is up to the individual parsers.
The DOM parser does almost consistently worse on the reduced files. This seems strange since the DOM structure is holding all tag information, and the tags require much less bytes to represent their names in the reduced files. This can mean one of two things, either there is a different cost not identified yet that overrides the cost of keeping the tags in the DOM-structure, or the cost of keeping the tags in the DOM structure is somehow relatively independent of the length of the tag names. This could happen if the tag names are stored as objects once in the DOM-structure and only links to the objects are passed around the structure.

10 Results

As we have seen from the statistical analysis, the null hypothesis that all XML takes equally long time could be discarded. The alternative hypotheses that the size of the XML has an effect on processing time and memory usage, and that the number of tags has an effect on processing time and memory usage was instead proven with a 5% margin of error.

Although both of these theories must be modified to only apply over a certain threshold which might be hardware or operating system dependent and also to only apply for SAX parsing

These alternative hypotheses only being valid above a certain threshold points towards there being a constant cost in setting up the parser that is independent from the file that is to be parsed. Only when the file gets big enough in size or tags does the cost exceed this constant cost

In addition, we have seen that reducing the redundancy in tags of XML can be an efficient way of shrinking the files by as much as 50%. For parsing with SAX, this also makes the files cheaper to parse. However when parsing with DOM it seems like the files gets more expensive to parse, at least with the method of executing the redundancy reduction used in this project.

11 Evaluation

11.1 Significance of results:

The results gained in this project can help the development of ways to extend the range of useful XML messages on weak devices.

A connection between size of the XML file and the amount of time and memory to needed parse it has been statistically proven with a margin of error of 5%.

A connection between the number of tags in XML files and the amount of time and memory needed to parse it has been statistically proven with a margin of error of 5.

This project has shown that reducing tag redundancies can lead to faster and more memory-efficient parsing of XML when using SAX parsers.

This project has also shown that reducing tag redundancies can lead to faster parsing of XML when using DOM, but no gain in the amount of memory needed was proven.

To summarize: Using a method to reduce redundancies in XML will lead to smaller XML files. These files are cheaper to parse and to transmit over wireless medias. When using DOM, a slight increase in cost in terms of memory was seen, but if this small increase can be handled, then using the redundancy reduction scheme proposed in this project will lead to faster parsing and cheaper transmitting.

11.2 Validity of results:

By doing the tests 1000 times on each file type, any inconsistencies in how Java behaves at different circumstances has been should have been effectively countered. Since for each test of parsing of a certain class of XML (different sizes, tag levels and tag number), a new random file was generated (within that class) on which the tests were run, any unforeseen variables which might bias the result has hopefully been countered. This however might not have happened if the program that generated the XML was consistent in putting such variables into the XML.

The only "fault" in the program that generates XML is that it generates very few tags that are the same when giving them random names, so it introduces very little tag repetition, which again leads to very little redundancy. This means that it was unable to provide the XML for the tests of hypothesis D. This lack of redundancy might have affected the other tests in some way, to which extent is unknown.

By using 3 SAX parsers for hypotheses A-C more credibility is given to the results found. Since the parsers for the most part agreed, the probability that the effects found are general enough to extend to most SAX parsers available is quite high.

By only using 1 DOM parser for testing the effect of tag redundancy reduction, the results gained needs to be read with a pinch of salt. It is very possible that the results are very specific for the Xerces DOM parser and different parsers would have given different results and even different conclusions. This is a weakness of this experiment. The only reason why more DOM parsers were not used was because of limitations in project time and availability of JAXP-adhering DOM parsers freely available for use.

12 Conclusion

Based on the results of the statistical analysis, the following 5 heuristic rules can be proposed for cheaper parsing of XML::

1: Try to keep the size of the XML file below a certain threshold.
In this project the threshold was found to be between 100 kB and 1 MB

2: Try to keep the number of tags below a certain threshold
In this project the threshold was found to be between 1000 and 10000 tags.

3: If communication costs are important, reduction of tags is a relatively simple means to reduce this by as much as 50% for some XML. This depends on the level of redundancy in the XML.

4: If using DOM to parse XML, redundancy-reduction can lead to more expensive XML to parse in terms of memory and time needed.

5: If using SAX to parse XML, redundancy-reduction can make the parsing cheaper both in terms of time and memory, but only if the XML is above a certain threshold size.
In this project this threshold was found to be between 465 Bytes and 4.95 kB.

13 Further work:

In this project, there were a couple of things that either fell a bit outside the focus or that there simply was not time for:

Testing hypothesis A-C with DOM parsers. It would be interesting to see how DOM parsers behave with different input as opposed to SAX parsers.

Other schemes for reducing costs of parsing XML than reducing redundancies might be conceivable. It would be interesting to see how different schemes would work in practice. From the results in this project, such a scheme would aim at either reducing the size of XML or the number of tags in the XML.

Taking this project and implementing it on some mobile devices to see how correct the finds done in this project are in regards to mobile devices. This project was done on a personal computer, and the finds are not necessarily directly transferable to mobile devices. Therefore it is a natural extension of this project to implement it again on a mobile platform.

Finding an expression for how much space is saved by doing various forms of redundancy reduction on different files. Sometimes there is huge savings to be made on the communication cost by reducing redundancies in the XML. Finding out when this is and how much can be saved is important to implementing optimal solutions.

.

14 Index:

XML – Extensible Markup Language, a markup language defined by the W3C, (World Wide Web Consortium)

SAX – Simple API for XML parsing, an API based on “push parsing”, i.e. the parser reads an XML file and sends events to the application using the parser as they are read.

DOM – Document Object Model, the original way of parsing XML as recommended by the W3C. all the data are first read into memory, then queries on the data can be made.

Tag Redundancy – In XML, tag names are repeated each time the tag is used, if this name is long it can lead to much wasted space.

J2ME – Java 2 Micro Edition; the version of the Java programming language meant for small and weak devices.

J2SE – Java 2 Standard Edition; the version of the Java programming language meant for workstations and laptops.

JAXP – Java API for XML Processing supports processing of XML documents using DOM, SAX, and XSLT enabling the parsers to be swapped without making changes to the application code.

Parser – A program that reads XML and gives an application program access to the information contained in the XML.

15 Bibliography

Books and papers:

[1] W. Rockwell, “XML, XSLT, Java, and JSP. A Case Study in Developing a Web Application” *New Riders.* ISBN: 0-757-1089-9

[2] N. Chase., “XML and Java from scratch”, *QUE,* ISBN: 0-7897-2476-6

[3] E.R. Harold & S. Means., “XML In a Nutshell, A desktop quick reference”, *O’Reilly,* ISBN 0-596-00058-8

[4] C. Wohlin et al. “Experimentation In Software Engineering, An Introduction”, *Kluwer Academic Publishers* , ISBN: 0-7923-8682-5

[5] Walpole et al. “Probability and Statistics for Engineers and Scientists, 6. ed.”, *Prentice Hall*, ISBN: 0-13-095246-X

[6] Gessler & Kotulla, “PDAs as mobile WWW browsers” *Telecooperation Office, University of Karlsruhe*, 1995

[7] Mascolo et al, “xmiddle: A Data-Sharing Middleware for Mobile Computing” *Dept. of Computer Science, University College London*, 2002

[8] White Paper on KVM and the Connected, Limited Device Configuration (CLDC), *Sun Microsystems* , http://java.sun.com/products/cldc/wp/KVMwp.pdf

Links to the internet:

[9] http://www.mowahs.com

[10] http://www.w3c.org

[11] http://www.sun.com

[13] http://xml.apache.org/

[12] http://piccolo.sourceforge.net/

[14] http://java.sun.com/xml/

[15] http://kobjects.dyndns.org/kobjects/auto?self=%23c0a80001000000f5ad6a6fb3

[16] http://xmlpull.org/

[17] http://java.sun.com/

Appendix A

kXML SAX parser

Hypothesis A - a	tags: 1	level: 1		
Size:	time mean	time var	mem mean	mem var
10	invalid			
100	240,75	1694,70	126857,95	3,52E+09
1000	245,15	1737,16	126299,26	3,23E+09
10000	283,10	2020,39	133210,18	2,70E+09
100000	386,16	3071,25	296887,97	2,40E+10

Hypothesis A - b	tags: 100	level: 4		
Size:	time mean	time var	mem mean	mem var
1000	249,06	1702,20	129228,33	3,09E+09
10000	294,94	1943,68	137168,03	2,13E+09
100000	399,83	2836,44	163812,18	1,71E+09
1000000	855,79	7715,98	748974,88	3,59E+09

Hypothesis B - a	size: 1k	level: 1		
tags:	time mean	time var	mem mean	mem var
1	246,88	1801,67	126779,00	3,32E+09
10	246,41	1848,29	129853,70	2,49E+09
100	250,03	1821,79	130846,22	2,58E+09

Hypothesis B - b	size: 100k	level: 4		
tags:	time mean	time var	mem mean	mem var
10	384,47	3450,50	163362,94	1,51E+09
100	401,77	3238,61	161180,26	1,54E+09
1000	459,23	3706,46	192178,50	1,82E+09
10000	983,70	6414,71	295304,72	8,74E+08

Hypothesis C - a:	size: 1k	tags: 100		
Avg. level:	time mean	time var	mem mean	mem var
2	251,29	2122,28	130141,93	3,19E+09
4	249,95	1986,68	126762,10	3,92E+09
8	253,76	2324,37	128386,92	3,25E+09
16	254,75	3236,93	130369,83	2,88E+09

Hypothesis C - b:	size: 100k	tags: 10k		
Avg. level:	time mean	time var	mem mean	mem var
4	1034,17	7405,06	328023,36	1,03E+09
8	1008,75	6551,65	304728,30	9,31E+08
16	1053,79	8288,68	346746,51	1,56E+09
32	1022,07	7144,01	303466,35	9,92E+08
64	1067,33	7153,81	352394,65	1,52E+09

Xerces SAX parser

Hypothesis A - a	tags: 1	level: 1		
Size:	time mean	time var	mem mean	mem var
10	invalid			
100	1415,04	21355,71	291360,14	3,59E+09
1000	1409,62	18776,64	296337,84	3,77E+09
10000	1445,36	18050,89	291866,00	3,75E+09
100000	1569,89	21169,63	294516,70	3,65E+09

Hypothesis A - b	tags: 100	level: 4		
Size:	time mean	time var	mem mean	mem var
1000	1420,37	19679,31	292613,14	3,36E+09
10000	1466,74	18219,13	292099,14	3,65E+09
100000	1595,68	21090,52	292142,03	3,63E+09
1000000	1984,26	25063,65	518242,50	8,99E+09

Hypothesis B - a	size: 1k	level: 1		
tags:	time mean	time var	mem mean	mem var
1	1407,71	17286,28	290628,81	3,61E+09
10	1410,65	18310,42	290832,78	3,72E+09
100	1406,38	18081,22	293364,21	3,52E+09

Hypothesis B - b	size: 100k	level: 4		
tags:	time mean	time var	mem mean	mem var
10	1579,29	22617,49	294361,90	3,54E+09
100	1593,01	20818,35	296557,99	4,02E+09
1000	1675,22	20893,63	298720,74	3,07E+09
10000	2006,17	22698,34	345308,99	1,61E+09

Hypothesis C - a	size: 1k	tags: 100		
level:	time mean	time var	mem mean	mem var
2	1425,66	17299,53	292370,71	3,40E+09
4	1420,20	18984,26	291311,80	3,39E+09
8	1423,88	17718,18	293607,58	3,47E+09
16	1424,99	17857,58	296654,46	3,79E+09

Hypothesis C - b	size: 100k	tags: 10k		
level:	time mean	time var	mem mean	mem var
4	2040,04	23479,33	343143,14	1,77E+09
8	2023,86	24578,06	349092,98	1,73E+09
16	2051,55	26875,64	348978,73	1,97E+09
32	2017,15	24461,69	354131,42	2,12E+09
64	2052,31	22578,59	364692,14	2,68E+09

Piccolo SAX parser

Hypothesis A - a	tags: 1	level: 1		
Size:	time mean	time var	mem mean	mem var
10	invalid			
100	838,51	19156,78	482051,06	4,71E+09
1000	839,57	18186,79	484737,14	4,89E+09
10000	889,05	18835,96	490084,61	3,69E+09
100000	991,56	20166,11	506861,10	1,84E+09

Hypothesis A - b	tags: 100	level: 4		
Size:	time mean	time var	mem mean	mem var
1000	854,35	17275,30	486565,75	4,52E+09
10000	917,62	20183,91	498953,11	3,00E+09
100000	1004,31	20758,91	517298,28	2,18E+09
1000000	1344,55	34301,94	820267,30	1,28E+10

Hypothesis B - a	size: 1k	level: 1		
tags:	time mean	time var	mem mean	mem var
1	839,56	16466,17	484201,02	4,19E+09
10	850,52	18028,73	484560,01	4,32E+09
100	856,82	18019,90	490315,57	3,84E+09

Hypothesis B - b	size: 100k	level: 4		
tags:	time mean	time var	mem mean	mem var
10	997,82	23381,70	509245,22	1,98E+09
100	997,58	28421,12	515741,14	2,04E+09
1000	1096,41	32839,81	550246,45	1,49E+09
10000	1629,88	68195,98	656444,04	2,23E+09

Hypothesis C - a	size: 1k	tags: 100		
level:	time mean	time var	mem mean	mem var
2	726,85	27720,50	490767,66	6,10E+09
4	732,55	21326,18	494973,90	5,01E+09
8	761,08	26163,12	493600,36	4,89E+09
16	728,44	32536,25	492157,69	6,79E+09

Hypothesis C - b	size: 100k	tags: 10k		
level:	time mean	time var	mem mean	mem var
4	1680,49	52618,57	653492,61	1,45E+09
8	1714,72	57278,08	654745,34	1,49E+09
16	1726,35	52636,67	661061,83	2,06E+09
32	1813,56	64848,65	664407,26	1,64E+09
64	1809,07	57675,66	670099,38	2,76E+09

Appendix B

This is a sample of the XML used in this project. The first is generated for testing of hypothesis A in test run b, the second is generated for testing of hypothesis D with 10 elements.

B.1 XML used in testing hypothesis A.

<start><r><yo><mhv><y><tg><isn><zgz></zgz></isn><dht></dht></tg><kz><hc></hc><fff></fff><bwp></bwp></kz><zxq><iww></iww></zxq><f></f><x><t></t></x></y><ege><up><ce></ce></up><ljh><ae>dvbuvzekpueaqmzosijjireuegjdpdivocldwgihbbjmgsowuqucwuaahuixjdfzhyojnnijqjhajkyidioiyrlfwmtlykjiiixriuby</ae></ljh></ege><box><kur><of></of></kur></box><wxk><c><c></c></c><ydb></ydb></wxk><y></y><m></m></mhv><xf><z><f><vfy></vfy><d></d></f><sv></sv></z><n></n></xf><xt><pqr></pqr><wcr></wcr></xt><e><fi></fi><wae>tlhejkxlregeupnpayzsjqussznlocjonjrdogjvkdgwketowanuknfxawuovvnsowyxzxbuyzwdqtqlkcy</wae><k></k></e><mif><pzs></pzs></mif><q><jlh></jlh></q><c></c></yo><as></as></r><vz><z><wvx></wvx></z></vz><qoh>jwpupbtyvsociaakmkjbvkeacnflcxvtzbvgfiozbkhctzjdibrlektsyuketrrttmjoel<ouv></ouv></qoh><an><aoa><k></k><fhy><x></x><r></r></fhy><r><xky></xky></r><hm></hm></aoa></an><m></m><tuh> <wwv><j></j></wwv> <okh><tlb></tlb></okh></tuh><lo></lo><peh></peh><u></u><aq></aq><rc></rc><iar><jkf><nbv></nbv></jkf><cd><g></g></cd></iar><m></m><ex><ms></ms></ex><gx></gx><q></q><ll></ll><a><w></w><ig></ig><mxj></mxj><ba></ba><ax></ax><mg></mg><v></v><kxx></kxx><wu></wu></start>

As can be seen very little tag repetition is found. This is why a different XML format was needed to test hypothesis D.

B.2 Original XML used in testing hypothesis D.

<hvite_sider><oppforing>SOME_COMPANY_NAME1<organisasjonsnummer>
65599711</organisasjonsnummer><adresse><gateadresse>SOME ADDRESS gate 2
</gateadresse><postadresse>SOME ADDRESS gate 3
</postadresse></adresse><telefonliste><telefonoppforing><telefonnummer>36808263
</telefonnummer><telefonbeskrivelse>TELEFONSVARER</telefonbeskrivelse>
</telefonoppforing></telefonliste><beskrivelse>DESCRIPTION OF A COMPANY
BLAHBLAHBLAH... WE MAKE 4</beskrivelse></oppforing></hvite_sider>

B.3 Reduced XML used in testing hypothesis D.

<root>

*<xshrinkheader><root>root</root>hvite_sider<d>organisasjonsnummer
</d><k>telefonbeskrivelse</k><h>telefonliste</h><j>telefonnummer</j>
<f>gateadresse</f><e>adresse</e>telefonoppforing<c>oppforing</c><g>
postadresse</g><l>beskrivelse</l></xshrinkheader>*

<c>SOME_COMPANY_NAME1<d>65599711</d><e><f>SOME ADDRESS gate
2</f><g>SOME ADDRESS gate3</g></e><h><j>36808263</j>
<k>TELEFONSVARER</k></h><l>DESCRIPTION OF A COMPANY
BLAHBLAHBLAH... WE MAKE 4</l></c>

</root>

This is the same XML as is seen in section B.2, but the tags have been reduced.
For ease of reading, the XML file has been split up into the root tag, the header, and the data. The header has been made italic. In the header all the necessary information to transform this xml back to its original form can be found. It is obvious that the data takes up less space in the reduced XML, however, the translating information makes this new file larger than the original. I.e. there is not enough redundancy in the XML to make this a worthwile excercise.

Appendix C

Overview of content from the cd rom: The folders are as follows:

-bin: Binary class files used throughout the project
-sample xml: One sample file for each of the classes of
XML used in the project.
-results: The outputted results from doing tests on XML
-statistics: The mean values and variances from the results.
-source: Source code to all binary files.
-jar: The .jar files needed to compile the source files.
-doc: The Javadoc documentation for this project
-attic: files that were made but for some reason were not
used to obtain the results of the project.
-report: This report in .doc format.

The Java classes used in this project are as follows:

-GenererXML and Node: Generates a random XML file and displays it on the screen.
Usage: "java GenererXML <avg.level> <size> <#tags> "
<avg. level> is the wanted average level of bytes in the XML
<size> is the wanted size of the XML
<#tags> is the wanted number of tags in the XML

-GenererHviteSiderXML: Generates XML of a form similar to the white or yellow pages in a phone book and writes it to a file.
Usage: "java GenererHviteSider <filename> <#posts>
<filename> is the name of the file to write to
<#posts> is the number of posts in the file.

-ParseTimer: This is the class that runs tests and writes results to file.
Usage: java ParseTimer <xml-file> <output> <parser>
<xml-file> is the file to be parsed
<output> is the file where the stats from the parsing is outputted
<parser> is a keyword for which parser is to be used. Valid keywords are: "xsax1", "ksax1", "picc1" and "xdom1"

-Memchecker: This class is used with the ParseTimer and takes measurements of the memory usage during the parsing.

-ParseKXML_SAX1, ParsePiccolo1, ParseXercesSAX1, ParseXercesDOM1:
These classes are all used by the ParseTimer to parse files with the kXML SAX parser, Piccolo SAX parser, Xerces SAX parser, Xerces DOM parser respectively

-ParseShrink, ParseXercesDOMShrink1: These classes are used to parse a file that has been tag reduced. The former with one of the SAX parsers, the latter with the DOM parser.

-Experiment1Handler: This class is used by the ParsePiccolo1 and ParseXercesSAX1 classes. It makes some statistics on the XML and can be seen as the application sofware for this experiment.

-SAXEventUnshrinkHandler: This class is used by the ParseShrink class to translate the tag names during parsing.

-XMLKrymper: Takes an XML file and executes tag reduction on it.
Usage: java XMLKrymper <xml file> <output>
<xml file> is the file to be reduced
<output> is the file to write the new reduced XML to.

-Statistic, Statistic2: These classes reads the first 1000 entries in the experiment logs outputted from the ParseTimer class. The former for testing hypothesis A-C, the latter for hypothesis D.

-MyParser: This is a helping class made to make itmore convenient to change between parser.

Appendix D