Final Thesis

Using XML for Import and Export of Data

by

Martin Axlid

LiTH-IDA-Ex-01/61

2001-06-08

Linköpings universitet
Department of Computer and Information Science

Final Thesis

Using XML for Import and Export of Data

by

Martin Axlid

LiTH-IDA-Ex-01/61

2001-06-08

Supervisor: Torbjörn Eriksson (Ericsson Radio Systems)
Examiner: Zebo Peng (Linköping University)

Abstract

Ericsson is developing a Corba service for data storage of radio networks. This service is implemented on top of an object database. The database contains data that describes a model of the physical network and its configuration. One task is to import and export the configuration data. Today XML is used as the file-format for the import and export. The current implementation of the import/export function has a linear growth of heap-memory consumption when the XML-files are processed. This causes the possibility of a fatal error when large amount of data should be handled. The purpose with the first part of the thesis has been to study and compare alternative XML-parsing techniques with limited memory consumption. The study shows that the best solution would be to use a combination of a SAX and DOM-parser in the import, and a non-standard “hardcoded” solution in the export.

Another task is to migrate data from one network model format to another; this is today performed outside the service. This can be very time-consuming, especially when the network model contains many elements, and there is therefore a need to make the process fully- or semi-automatic. The purpose of the thesis’s second part has been to find a suitable technique to perform the conversion. The study shows that an implementation of a new conversion tool in Java will be most effective and flexible. The use of a standard XML-conversion technique like XSL or a third party product would be less effective.

There is a need to make the format of the XML-file as effective as possible with respect to the following factors: correct functionality, easy implementation, simple readability and good runtime performance. In the third part of the thesis, the current format has been compared to several other “standard” XML-formats. The conclusion of this study is that the other formats do not have any significant advantage over the current format. The best solution would be to apply some minor changes to the current format and continue to use that.

Table of Contents

1 Introduction

This chapter contains information about the background, purpose and basic concepts of the thesis.

1.1 Background and Purpose

Ericsson is developing a CORBA service for data storage of radio networks. This component is called Configuration Service (CS). The service is implemented on top of an object database. The database contains data that describes a model of the physical network and its configuration. The service includes methods for import and export of configuration data. Ericsson is today using XML as the file-format for import and export.

The purpose of the thesis is to study how different sections of the import and export function could be improved. One part is to study how large import and export files should be handled to avoid memory and performance problems. Furthermore, there is a need to make the migration between different network-models fully- or semi-automatic. Different techniques for comparison and conversion of XML-files should be evaluated. Finally, the format of the XML-file should be studied to see if any changes could increase the performance.

1.2 Target Group

Since the problems mainly are referred to the import/export functionality in the CS-component, this report is intended to be read by people with knowledge of the CS in general and the import/export function in particular. However, people with a general interest in XML-technology can also benefit from the information in some of the sections.

It is assumed that the reader of the document has basic knowledge of object-oriented programming and design.

1.3 Definitions

Common Object Request Broker Architecture (CORBA)

CORBA is OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. (OMG 2001)

Directory Service

A directory service is a specialised database that is read or searched far more often than it is written to. It supports storing a wide variety of information and provides a mechanism to extend the types of information that can be stored. Directory services can be centralized or distributed. They are often distributed in large scale, both in how and where information is distributed. Directory services are usually replicated so that they are highly available to the clients accessing them.

Distinguished Name (DN)

A DN is used to uniquely identify a MO within a MIB (for information about MO and MIB, see section 2.2 and 2.3). It is built from a series of “name components” denoting a containment hierarchy.

Extensible Stylesheet Language (XSL)

XSL is a specification intended to be used for transforming XML documents into other formats, e.g. HTML or a different formatted XML document.

Java Virtual Machine (JVM)

The Java Virtual Machine is an abstract computing machine. Like a real computing machine, it has an instruction set and uses various memory areas. It is reasonably common to implement a programming language using a virtual machine. (SUN 2001)

Lightweight Directory Access Protocol (LDAP)

LDAP is, like X.500, both an information model and a protocol for querying and manipulating it. LDAP's overall data and namespace model is essentially that of X.500. The major difference is that the LDAP protocol itself is designed to run directly over the TCP/IP stack, and it lacks some of the more esoteric DAP protocol functions. (KMS 2001)

Namespaces

An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set. (Sun 2001)

The namespace standard lets you write an XML document that uses two or more sets of XML tags in modular fashion. Suppose for example that you created an XML-based parts list that uses XML descriptions of parts supplied by other manufacturers. The "price" data supplied by the sub-components would be amounts you want to total up, while the "price" data for the structure, as a whole, would be something you want to display. The namespace specification defines mechanisms for qualifying the names so as to eliminate ambiguity. That lets you write programs that use information from other sources and do the right things with it. (Sun 2001)

World Wide Web Consortium (W3C)

The World Wide Web Consortium was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has more than 500 Member organisations from around the world and has earned international recognition for its contributions to the growth of the Web.

X.500

X.500 is an overall model for Directory Services in the OSI world. The model encompasses the overall namespace and the protocol for querying and updating it. The protocol is known as "DAP" (Directory Access Protocol). DAP runs over the OSI network protocol stack -- that, combined with its very rich data model and operation set makes it quite "heavyweight". It is rather tough to implement a full-blown DAP client and have it "fit" on smaller computer systems. Thus, the folks at University of Michigan, with help from the ISODE Consortium, designed and developed. (KMS 2001)

1.4 Readers Guidelines

This section contains a short description of each chapter.

Chapter 1, Introduction: contains information about the background, purpose and definitions used in the thesis.

Chapter 2, Configuration Service Overview: contains information about the basic concepts of the Configuration Service component.

Chapter 3, Basic Theory About XML and Parsers: contains information about the basic concepts of XML and related topics.

Chapter 4, Large Import and Export Files: contains information about the problems that occurs when large amount of data is being imported or exported from the database.

Chapter 5, Migrating Data Between Different MIM-versions: contains information about how to make the migration between two different MIM-versions fully- or semi-automatic.

Chapter 6, DTD/Schema Format: contains information about which DTD/schema format that is most effective to use for the MIB XML-file.

Chapter7, Summary and Future Work: contains a summary of the results and conclusions in the thesis. Some suggestions about how the process should continue are also included.

Chapter 8, Acronyms and Abbreviations: specifies the acronyms and abbreviations that are used in the thesis.

Chapter 9, References: specifies the references that are used in the thesis.

2 Configuration Service Overview

This chapter contains information about the basic concepts of the Configuration Service component.

2.1 Introduction

The Configuration Service enables a user to configure a radio network such as GSM or UMTS. A user may be some application in the GSM OSS or RANOS, or an external Network Management System. The user-applications access the CS functions through some supplied CORBA IDLs.

Most components are implemented in Java, which means that some of the problems that are studied in the thesis are related to the Java language.

Some of the main concepts in the Configuration Service are: Managed Information Model (MIM), Managed Information Base (MIB), Managed Object (MO) and relations.

2.2 Managed Information Model

Managed Information Model (MIM) is the information model. It describes the managed objects that can exist and the relationships between them. In database connection it could be the database schema. In Object Oriented programming it would be the collection of classes and the relations between them. (CS 2000)

Figure 1 gives an example of a MIM for a network in GSM.

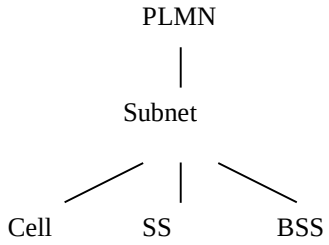

Figure 1: MIM for a subnetwork

2.3 Managed Information Base

Managed Information Base (MIB) is an instance of a MIM. The purpose of the MIB is to define what MIM model the different MO's shall follow. A MIB instance of the MIM in Figure 1 shall contain a root MO of class type PLMN, below it a MO of class type Subnet and so on. (CS 2000)

2.4 Managed Object

Managed Object (MO) can be seen as an object whose class definition can be found in the MIM. A MO contains zero or more attributes. (CS 2000)

2.5 Relations

Relations between MO's can be of two kinds, containment relations and association relations. (CS 2000)

An association relation between two MOs in different MIBs is called a hopper. Figure 2 shows some examples of the different CS relation types.

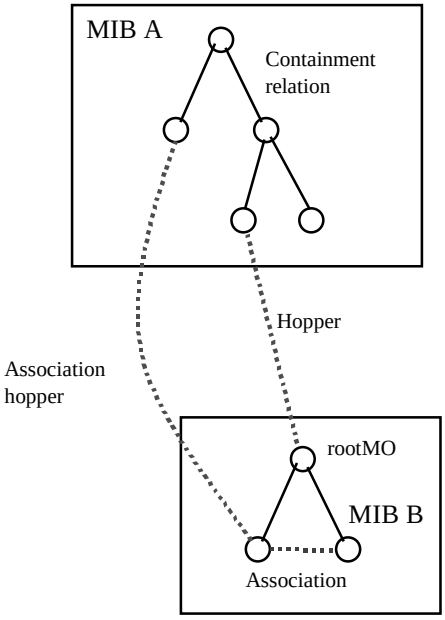

Figure 2: CS Relations

2.5.1 Containment Relations

Containment relations within a MIB is always automatically created and sustained when an object is created. It is a parent-child relation that is visible through the distinguished name of a managed object. If the parent has the name a,b,c then children $d_1..d_n$ are named a,b,c,d_1 ... a,b,c,d_n.

Containment relations also connects different MIBs. The rule is the same here; the parent-child relation must also be visible in the name hierarchy.

2.5.2 Association Relations

Association relations are used to describe side relations between MO's. Associations can be between MOs within the same MIB, or between MOs in different MIBs. They are created and deleted with different methods in the CS interface.

2.6 Example

In Figure 3 we have a small network with four different MIBs based on three different MIMs, The Subnetwork MIM, the MSC MIM and the BSC MIB. There are containment relations from SS node to MSC node and from BSS nodes to BSC nodes. The full distinguished name for the MSC will be the prefix + its own name that is *PLMN=plmn,Subnet=Subnet,SS=SS1,MSC=MSC1*. There are also associations between Cell to InnerCells. (CS 2000)

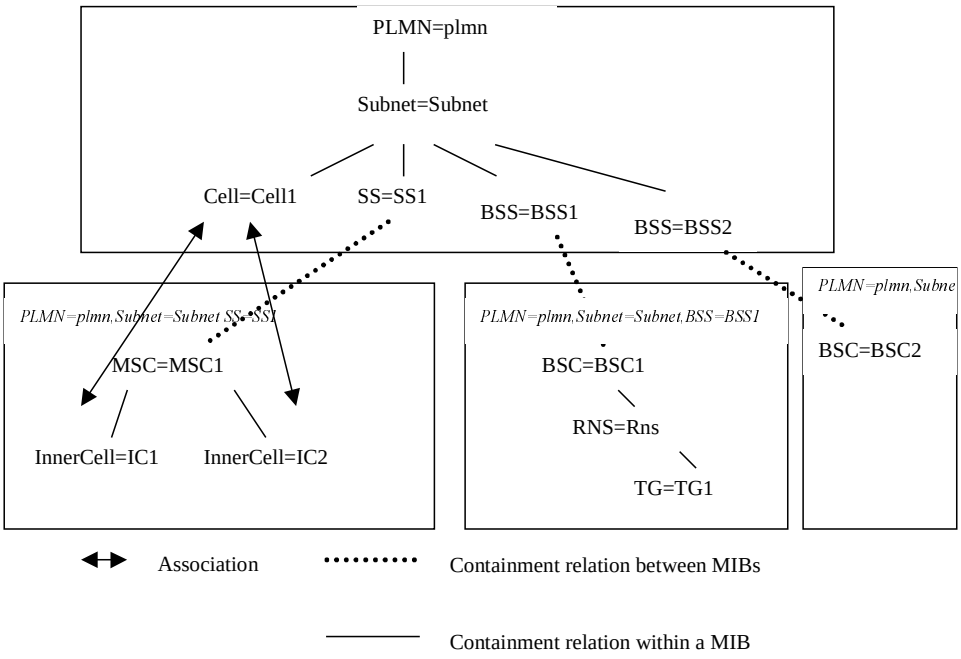

Figure 3: Example of a small network

The different MIBs can be in one CS or in different CS. A MIB can never be split between different CS. (CS 2000)

3 Basic Theory About XML and Parsers

This chapter contains information about the basic concepts of XML and related topics.

3.1 XML Overview

XML is an abbreviation for eXtended Markup Language. It is a text-based mark-up language that is fast becoming the standard for data interchange on the Web. As with HTML, you identify data using tags (identifiers enclosed in angle brackets like this: `<...>`). Collectively, the tags are known as "markup". (Sun 2001)

But unlike HTML, XML tags tell you what the data means, rather than how to display it. Where an HTML tag says something like "display this data in bold font" (`<b>...</b>`), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: `<message>...</message>`). (Sun 2001)

In the same way that you define the field names for a data structure, you are free to use any XML tags that make sense for a given application. Naturally, though, for multiple applications to use the same XML data, they have to agree on the tag names they intend to use. (Sun 2001)

Here is an example of some XML data you might use for a messaging application:

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>XML Is Really Useful</subject>
    <text>
       How many ways is XML useful? Let me count...
    </text>
</message>
```

The tags in this example identify the message as a whole, the destination and sender addresses, the subject, and the text of the message. As in HTML, the `<to>` tag has a matching end tag: `</to>`. The data between the tag and its matching end tag defines an element of the XML data. Note, too, that the content of the `<to>` tag is entirely contained within the scope of the `<message>...</message>` tag. It is this ability for one tag to contain others that gives XML its ability to represent hierarchical data structures. (Sun 2001)

Whitespace is essentially irrelevant, so you can format the data for readability and yet still process it easily with a program. Unlike HTML, however, you could easily search a data set for messages containing "useful" in the subject, because the XML tags identify the content of the data, rather than specifying its representation. (Sun 2001)

3.1.1 Tags and Attributes

Tags can also contain attributes -- additional information included as part of the tag itself, within the tag's angle brackets. The following example shows an email message structure that uses attributes for the "to", "from", and "subject" fields:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Useful">
    <text>
       How many ways is XML useful? Let me count...
    </text>
</message>
```

As in HTML, the attribute name is followed by an equal sign and the attribute value, and multiple attributes are separated by spaces. Unlike HTML, however, commas between attributes are not ignored -- if present, they generate an error. (Sun 2001)

3.1.1.1 Empty Tags

One really big difference between XML and HTML is that an XML document is always constrained to be well formed. There are several rules that determine when a document is well formed, but one of the most important is that every tag has a closing tag. So, in XML, the `</to>` tag is not optional. The `<to>` element is never terminated by any tag other than `</to>`. (Sun 2001)

Another important aspect of a well-formed document is that all tags are completely nested. So you can have
`<message>..<to>..</to>..</message>`, but never
`<message>..<to>..</message>..</to>`. (Sun 2001)

Sometimes, though, it makes sense to have a tag that stands by itself. For example, you might want to add a "flag" tag that marks message as important. A tag like that doesn't enclose any content, so it's known as an "empty tag". You can create an empty tag by ending it with `/>` instead of `>`. For example, the following message contains such a tag:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Useful">
    <flag/>
    <text>
       How many ways is XML useful? Let me count...
    </text>
</message>
```

(Sun 2001)

3.1.1.2 Comments in XML Files

XML comments look just like HTML comments:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Useful">
    <!-- This is a comment -->
    <text>
       How many ways is XML useful? Let me count...
    </text>
</message>
```

(Sun 2001)

3.1.1.3 The XML Prologue

A XML file always starts with a prologue. The minimal prologue contains a declaration that identifies the document as an XML document, like this:

```
<?xml version="1.0"?>
```

The declaration may also contain additional information, like this:

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="yes"?>
```

The XML declaration is essentially the same as the HTML header, `<html>`, except that it uses `<?..?>` and it may contain the following attributes:

version	Identifies the version of the XML markup language used in the data. This attribute is not optional.
encoding	Identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8.)
standalone	Tells whether or not this document references an external entity or an external data type specification. If there are no external references, then "yes" is appropriate

The prologue can also contain definitions of entities (items that are inserted when you reference them from within the document) and specifications that tell which tags are valid in the document. Both are declared in a Document Type Definition (DTD) that can be defined directly within the prologue, as well as with pointers to external specification files. (Sun 2001)

The declaration is actually optional. But it's a good idea to include it whenever you create an XML file. The declaration should have the version number, at a minimum, and ideally the encoding as well. That standard simplifies things if the XML standard is extended in the future, and if the data ever needs to be localised for different geographical regions. (Sun 2001)

3.1.2 DTD/Schema Validation

A DTD specifies the kinds of tags that can be included in your XML document, and the valid arrangements of those tags. You can use the DTD to make sure you don't create an invalid XML structure. You can also use it to make sure that the XML structure you are reading (or that got sent over the net) is indeed valid. (Sun 2001)

Below is an example of a DTD-definition:

```
<!ELEMENT message (subject, flag?, text)>
  <!ATTLIST message to CDATA #REQUIRED from CDATA
#REQUIRED>
 <!ELEMENT subject (#PCDATA)>
 <!ELEMENT flag EMPTY>
 <!ELEMENT text (#PCDATA)>
```

There are two different levels of an XML-file format. "Well Formed" XML documents are documents that conforms to the basic XML syntax rules, e.g. no overlapping element-definitions etc. "Valid" XML documents are "Well Formed" XML documents, which also conforms to the rules of a specified DTD.

The DTD specification is actually part of the XML specification, rather than a separate entity. On the other hand, it is optional - you can write an XML document without it. And there are a number of schema proposals that offer more flexible alternatives. So it is treated here as though it were a separate specification. (Sun 2001)

It is difficult to specify a DTD for a complex document in such a way that it prevents all invalid combinations and allows all the valid ones. So constructing a DTD is something of an art. The DTD can exist at the front of the document, as part of the prologue. It can also exist as a separate entity, or it can be split between the document prologue and one or more additional entities. (Sun 2001)

While the DTD mechanism was the first method defined for specifying valid document structure, it was not the last. Several newer schema specifications have been devised. However, DTD is the only schema that will be studied within this thesis.

3.1.3 Why Is XML Important?

There are a number of reasons for XML's surging acceptance. This section lists a few of the most prominent.

3.1.3.1 Plain Text

Since XML is not a binary format, you can create and edit files with anything from a standard text editor to a visual development environment. That makes it easy to debug your programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides scalability for anything from small configuration files to a company-wide data repository. (Sun 2001)

3.1.3.2 Data Identification

XML tells you what kind of data you have, not how to display it. Because the markup tags identify the information and break up the data into parts, an email program can process it, a search program can look for messages sent to particular people, and an address book can extract the address information from the rest of the message. In short, because the different parts of the information have been identified, they can be used in different ways by different applications. (Sun 2001)

3.1.3.3 Stylability

When display is important, the stylesheet standard, XSL, lets you dictate how to portray the data. For example, the stylesheet for:

```
<to>you@yourAddress.com</to>
```

can say:

```
Start a new line.
Display "To:" in bold, followed by a space
Display the destination data.
```

Which produces:

```
To: you@yourAddress
```

Of course, you could have done the same thing in HTML, but you wouldn't be able to process the data with search programs and address-extraction programs and the like. More importantly, since XML is inherently style-free, you can use a completely different stylesheet to produce output in postscript, TEX, PDF, or some new format that hasn't even been invented yet. That flexibility amounts to what one author described as "future-proofing" your information. The XML documents you author today can be used in future document-delivery systems that haven't even been imagined yet. (Sun 2001)

3.1.3.4 Inline Reusability

One of the nicer aspects of XML documents is that they can be composed from separate entities. You can do that with HTML, but only by linking to other documents. Unlike HTML, XML entities can be included "in line" in a document. The included sections look like a normal part of the document -- you can search the whole document at one time or download it in one piece. That lets you modularize your documents without resorting to links. You can single-source a section so that an edit to it is reflected everywhere the section is used, and yet a document composed from such pieces looks for all the world like a one-piece document. (Sun 2001)

3.1.3.5 Easily Processed

As mentioned earlier, regular and consistent notation makes it easier to build a program to process XML data. For example, in HTML a `<dt>` tag can be delimited by `</dt>`, another `<dt>`, `<dd>`, or `</dl>`. That makes for some difficult programming. But in XML, the `<dt>` tag must always have a `</dt>` terminator, or else it will be defined as a `<dt/>` tag. That restriction is a critical part of the constraints that make an XML document well formed. (Otherwise, the XML parser won't be able to read the data.) Since XML is a vendor-neutral standard, you can choose among several XML parsers, any one of which takes the work out of processing XML data. (Sun 2001)

3.1.3.6 Hierarchical

XML documents benefit from their hierarchical structure. Hierarchical document structures are, in general, faster to access because you can drill down to the part you need, like stepping through a table of contents. They are also easier to rearrange, because each piece is delimited. In a document, for example, you could move a heading to a new location and drag everything under it along with the heading, instead of having to page down to make a selection, cut, and then paste the selection into a new location. (Sun 2001)

3.1.3.7 Binding

Once you have defined the structure of XML data using either a DTD or the one of the schema standards, a large part of the processing you need to do has already been defined. For example, if the schema says that the text data in a `<date>` element must follow one of the recognized date formats, then one aspect of the validation criteria for the data has been defined -- it only remains to write the code. Although a DTD specification cannot go the same level of detail, a DTD (like a schema) provides a grammar that tells which data structures can occur, in what sequences. That specification tells you how to write the high-level code that processes the data elements. (Sun 2001)

But when the data structure (and possibly format) is fully specified, the code you need to process it can just as easily be generated automatically. That process is known as binding -- creating classes that recognize and process different data elements by processing the specification that defines those elements. As time goes on, you should find that you are using the data specification to generate significant chunks of code, so you can focus on the programming that is unique to your application. (Sun 2001)

3.1.4 Summary

XML is pretty simple, and very flexible. It is providing a common language that different computer systems can use to exchange data with one another. (Sun 2001)

3.2 XSL

The XML standard specifies how to identify data, not how to display it. HTML, on the other hand, told how things should be displayed without identifying what they were. The coalescing XSL standard is essentially a translation mechanism that lets you specify what to convert an XML tag into so that it can be displayed -- for example, in HTML. Different XSL formats can then be used to display the same data in different ways, for different uses. (Sun 2001)

Two important parts of XSL are XSL Transformations (XSLT) and XML Path language (XPath). XSLT is a language that defines how the transformation should be done. XPath is a language that defines which elements that should be included in the transformation. The following code shows an example of a XSL “match”-definition:

```
<xsl:template match="book">
  <fo:block>
    <xsl:apply-templates select=".//heading"/>
  </fo:block>
</xsl:template>
```

The “`.//heading`”-parameter is a XPath-statement that selects the heading elements in all the descending levels of the XML-file. The rest of the code are XSLT-statements.

A special XSL-transformation tool is needed to perform the actual transformation. There are several such tools available, e.g. Xalan. An overview of the XSL transformation concept is shown in Figure 4.

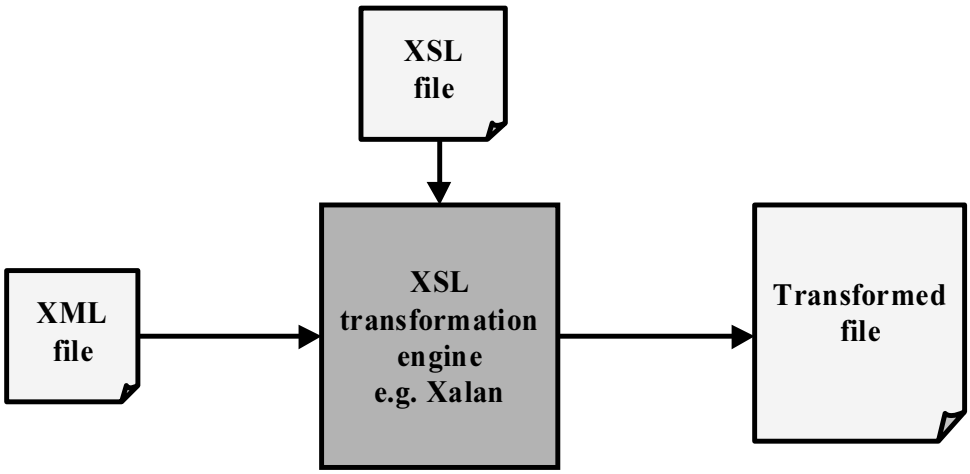

Figure 4: XSL transformation overview

It is possible to read data from several different XML-documents during a transformation, but usually only one document is used.

3.3 Parsers

The purpose of an XML-parser is to read, validate and access the elements in an XML-file. The two most common techniques for parsing XML-files are "Simple API" for XML (SAX) and Document Object Model (DOM).

SAX is an event-driven, serial-access mechanism that does element-by-element processing. To use a SAX parser, some specified callback methods has to be written. Those methods are then invoked from the parser whenever it encounters a XML tag (or encounters an error, or wants to tell you anything else). This design makes the SAX-parser fast with low memory-consumption, but will usually require more code at the application side.

The DOM API is generally an easier API to use. It provides a relatively familiar tree structure of objects. You can use the DOM API to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive. For that reason, the SAX API will tend to be preferred for server-side applications and data filters that do not require an in-memory representation of the data.

3.3.1 SAX Details

The SAX parser is a result from a discussion in the XML-DEV public mailing list 1997-1998. The basic idea of the project was to find a solution to the needless incompatibility of the different parsers available at that time.

A SAX-parser does not store any XML-data internally. The implementation of the different user-specified callback methods decides if, and how, any data should be saved. The parser has to be configured with the callback methods before the parsing of the XML-file can begin. The different callback methods are grouped into different handlers as seen in Figure 5.

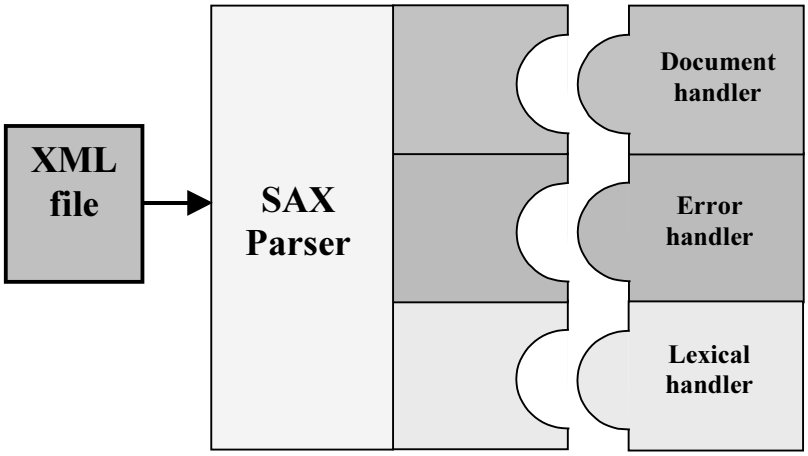

Figure 5: SAX Concept overview

Parser	The org.xml.sax.Parser interface defines methods like `setDocumentHandler`, to set up event handlers, and `parse`, to actually do the parsing.
DocumentHandler	Methods like `startDocument`, `endDocument`, `startElement`, and `endElement` are invoked when an XML tag is recognized. This interface also defines the methods `processingInstruction` and `characters`, which are invoked when the parser encounters an inline processing instruction or the text of an XML element, respectively.
ErrorHandler	Methods error, fatalError, and warning are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.
LexicalHandler	This handler include methods for `startDTD`, `endDTD` and other miscellaneous parts of the XML-file.

When a SAX-parser should be used within an application, the following steps has to be performed by the application:

- Create an instance of a class that implements the SAX-Parser interface
- Create a document-handler with the desired methods for handling the callbacks
- Configure the parser-instance with the document handler
- Start the parsing process by calling the `parse()`-method in the Parser-instance.

Figure 6 shows an interaction-diagram of the different steps. If any of the other types of SAX- handlers are needed, the procedure is the same as for the document-handler.

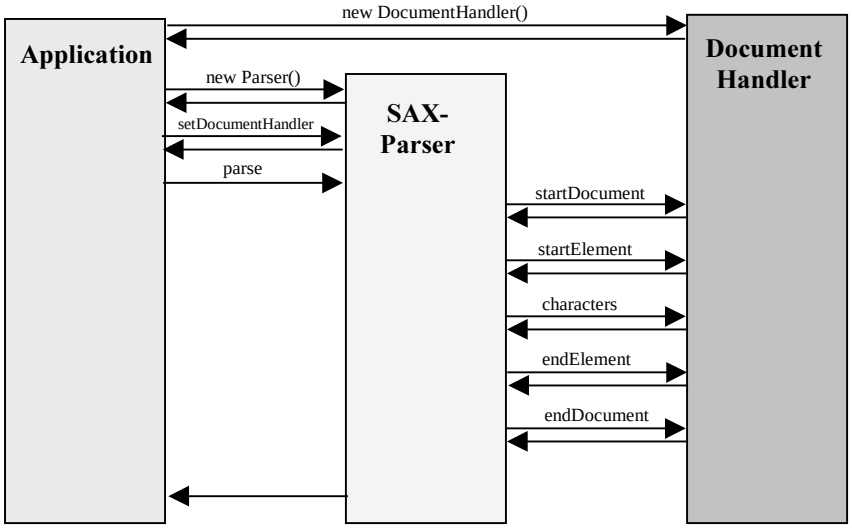

Figure 6: SAX-parser details (Birbeck, Mark et.al. 2000)

3.3.2 DOM Details

The DOM specification defines the Document Object Model, a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web. (W3C 2001)

DOM is being designed at several levels:

- Level 1. This concentrates on the actual core, HTML, and XML document models. It contains functionality for document navigation and manipulation.
- Level 2. Includes a style sheet object model, and defines functionality for manipulating the style information attached to a document. It also enables traversals on the document, defines an event model and provides support for XML namespaces.
- Level 3. Will address document loading and saving, as well as content models (such as DTDs and schemas) with document validation support. In addition, it will also address document views and formatting, key events and event groups. First public working drafts are available.
- Further Levels. These may specify some interface with the possibly underlying window system, including some ways to prompt the user. They may also contain a query language interface, and address multithreading and synchronisation, security, and repository. (W3C 2001)

A DOM data-tree document is built by means of a parser/builder function. When the DOM document has been produced, methods included in the DOM-specification can be used to find, add, alter and delete data or nodes in the tree.

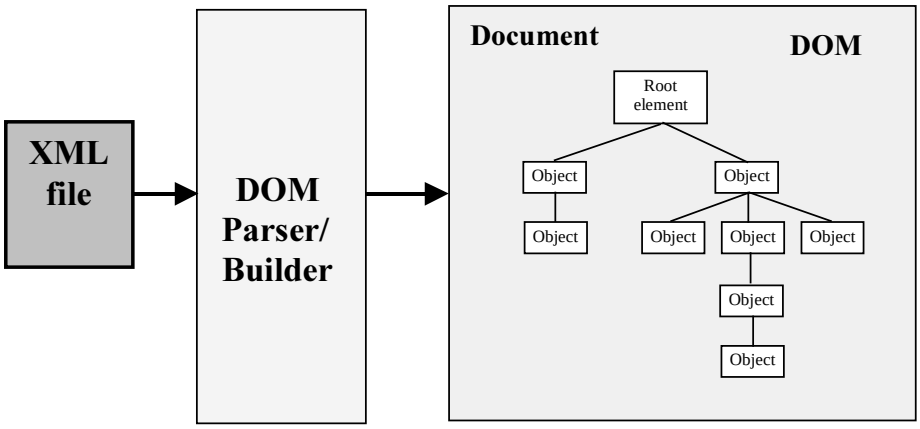

Figure 7: DOM concept overview

Most implementations store the data-tree in primary memory, but other storage-techniques are possible to use, e.g. databases or binary files.

4 Large Import and Export Files

This chapter contains information about the problems associated with the handling of large import and export configuration data files. Some of the subjects addressed are memory handling in JVM, evaluation of alternate methods, prototype design, test results and proposed solutions.

4.1 Problem Description

The Configuration Service component contains functions for import and export of configuration data to/from the database. Today XML is used as the file-format for the import and export. The current implementation of the import/export function has a heap-memory consumption that is linear to the size of the XML-file. This will cause a primary memory shortage problem when a network with a large number of elements should be configured/simulated. To solve this problem, the current methods must be based on a different technique.

4.2 Memory Configuration in Java

The memory available for a Java-program is not only dependent on the physical and virtual memory in the computer-environment; it is also dependent on the configuration of the JVM.

The Sun JVMs have a minimum and maximum heap size, that is configurable through command line parameters. Those parameters exists in order to give the system administrator control over how much memory resources the JVM will consume. This is important in production environments. The JVM will attempt to get a heap up to the maximum you have set from the operating system. In an OS that supports swapping, you can set the max larger than the physical memory available and the system will swap underneath you to make it available. In Java2 the default maximum heap size is usually 64 MB. The current absolute maximum Java heap size is about 2 GB (due to internals having to do with addressing.)

4.3 Import

The current import method is based on an in-memory DOM-parser. The other of the two common parsing standards, SAX, has a “serial” mechanism that makes the memory requirements for its data-structure negligible. If only this characteristic was considered, this technique should then have been a good choice. However, there are some drawbacks of using the SAX-parser. Since the SAX-parser itself does not store any data, algorithms have to be written that continuously store the data that is needed later on. The current level and state in the parsing has also to be continuously tracked.

The current methods for handling the radio network elements and their datatypes are based on the possibility to query the data from the created DOM-tree. To rewrite all those methods will be very time-consuming. To find a good relation between implementation time and the requirements of low memory consumption, a combination of the SAX and DOM techniques seems to be a good choice. The SAX should then be used as the main parser and smaller DOM-elements should be created to be able to call the most complex methods implemented in the current system, e.g. the attribute datatypes handling.

4.3.1 SAX Import Prototype Design

A prototype program has been written to prove the concept of using SAX as the main parser in the import method. The prototype simulates the database access by outputting the parameters needed in the database-transactions as text.

4.3.1.1 Class-Diagram

The design of the program is made in such a way that all significant functionality is included in the same class; the CSSAXTest-class includes the main-function together with the necessary callback functions. This means that the DefaultHandler is created by instantiating the same class.

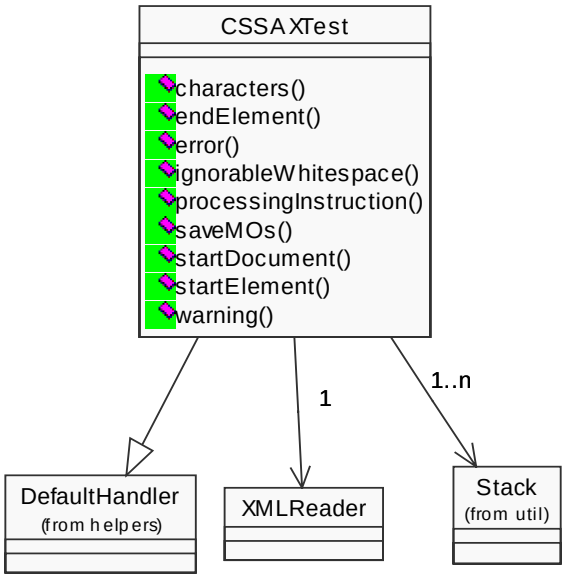

Figure 8: Import with SAX; class-diagram.

The prototype should work with any SAX2-compliant parser implementation. In this test the Xerces parser was used. A data-stack was chosen for the storage of the MO-parent hierarchy.

4.3.1.2 State-diagram

Due to the “Serial” mechanism of the SAX-parser, the application has to track the current position within the different types of elements in the MIB XML-file.

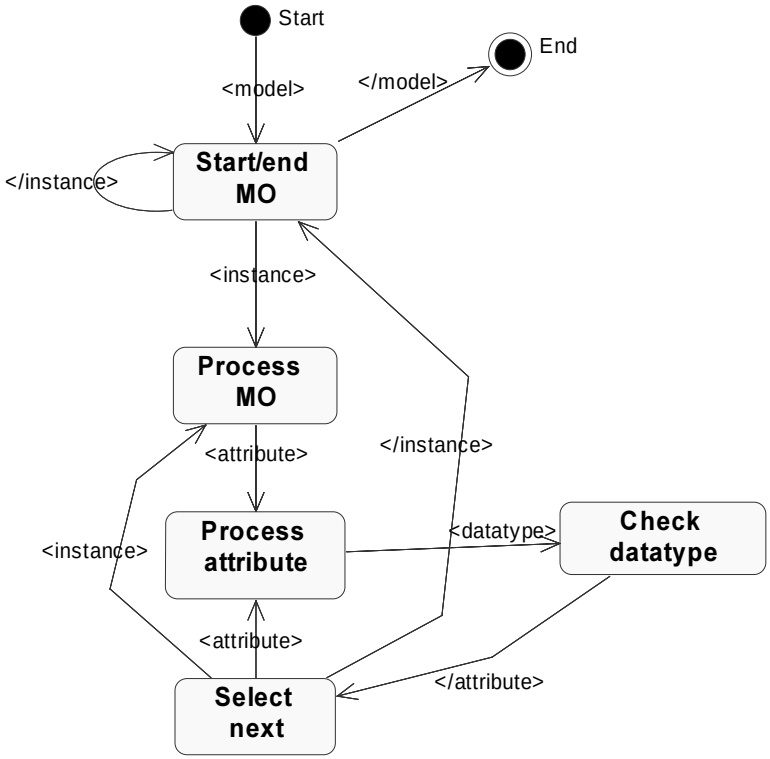

Figure 9: State diagram for SAX-import

Start/end MO	This State is entered from the start-state when a model tag is received in the startElement-method. When a model-end tag is received the MO-parsing is finished and the State is changed to End. The State is also entered when an instance-end tag is received in either state “Select next” or in its own state.
Process MO	This State is entered when an instance tag is received in either State “Start/end MO” or “Select next”.
Process attribute	This State is entered when a MO-attribute tag is received in either State “Process MO” or “Select next”.
Check datatype	This State is entered when a datatype tag is received in state “Process attribute”.
Select next	This State is entered when a MO-attribute end-tag is received in State “Check datatype”.

4.3.1.3 Test Result

The prototype tests show that there are no significant problems when the SAX-technique is used as the main parser. However, there are some things that should be considered, e.g. that the application is accessing the XML-elements before the whole file has been read. To ensure that no fatal validating errors occurs when only part of the model has been stored, the import must then start with a “dummy” parsing of the whole XML-file.

4.3.2 DOM Alternatives

As mentioned in section 3.3.2, there is nothing in the DOM-specification that states that the DOM data-tree has to be stored in the primary memory. The simplest solution is however to store the whole tree in primary memory and therefore most DOM-implementations have selected this alternative. To construct a parser with an alternative storage solution from scratch will be very time consuming and is objectionable in the perspective of the CS-development. There are, however, a few third party alternative implementations available, which have chosen to use either a database or a binary file as part of the tree-storage. The primary memory consumption in those implementations will be significantly lower, but at the cost of a slower runtime performance. The runtime performance loss is more significant in the solutions based on database-access. The database engine itself will also consume some primary memory, which is another drawback. The solution based on binary files has the qualification to produce the best memory/runtime ratio performance. To this date there is only one such parser available: the PDOM component included in the Infonyte XQL Suite which is based on the GMD-IPSI XQL Engine project (see Appendix A: PDOM). The PDOM-implementation has support for using a SAX-parser when building the PDOM-tree and it uses an advanced cache technique when queering the tree-elements.

4.4 Export

The current export method is also using an in-memory DOM. The main reason for this is that it is easier to format an XML-file when the data is read from a DOM-tree instead of the database. When a network-model in the database should be exported, the configuration data is transferred to a DOM-tree. Querying the nodes in the DOM in the correct sequence then creates the XML-file. However, this implementation creates the same memory problem as in the import function.

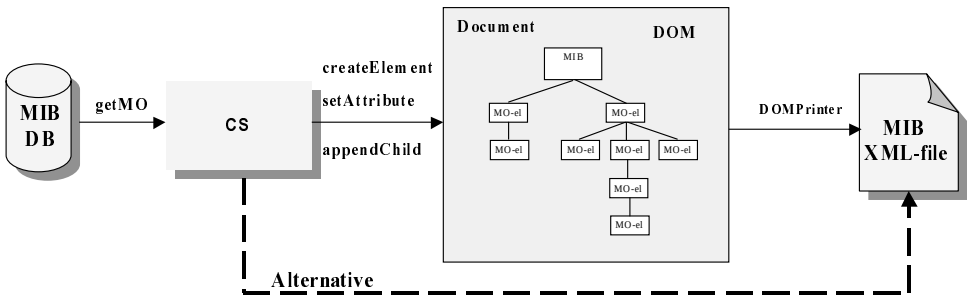

Figure 10: Export overview

If a solution to this should be found, the SAX-technique is no alternative because it is only designed to read XML-files. There are, as mentioned in section 3.3.1, no support for any internal data-structure or XML-file output formatting.

A solution that does not require the creation of a DOM-tree is to directly output the configuration data to an XML-file when it is read from the database. This will make the memory consumption negligible. To get a correctly formatted XML-file, code must then be added to keep track of the current level and state within the data-model. The elements also have to be read in a correct sequence from the database since the output is “serial”.

Another soulution could be to replace the current DOM with the PDOM-technique. It will have the same advantages as in the import-method, i.e. this would be the soulution that is easiest and fastest to implement in the current system.

4.5 Memory Consumption Comparison

Below is a compilation of several tests that were made to determine the memory consumption of different parser types.

The Xerces DOM parser is a conventional parser that stores the data-tree in primary memory. The tests were performed without node expansion and with validation.

The tests of the PDOM-parser were performed an evaluation version. According to the responsible company, their commercial version has a lot of improvements. The PDOM parser is, as earlier mentioned, based on a technique that stores the data-tree in a binary file. When the PDOM-document tree is built the callback-functions of a third party SAX-parser is used. In this test the PDOM-parser was configured to use the IBM xml4j2 SAX-parser.

The third parser tested was Xerces SAX-parser. The tests were performed with validation.

Filesize (MB)	NR of MOs	Xerces DOM	PDOM	Xerces SAX[1]
3.32	1006	17.03	4.95	0.78
6.67	2030	33.73	4.96	0.78
13.19	4038	63.90	5.04	0.78
180.86	48456	926.47[2]	12.09	0.78

Table 1: Parser memory comparison

The table shows how much of the heap-memory that is needed during the parsing. The values are in MB.

[1] The Heap-memory needed by the SAX-parser is in this case negligible and has therefore been estimated to a low value.
[2] Due to lack of enough primary memory in the test-environment, the last test-value is an estimate based on the previous measurements.

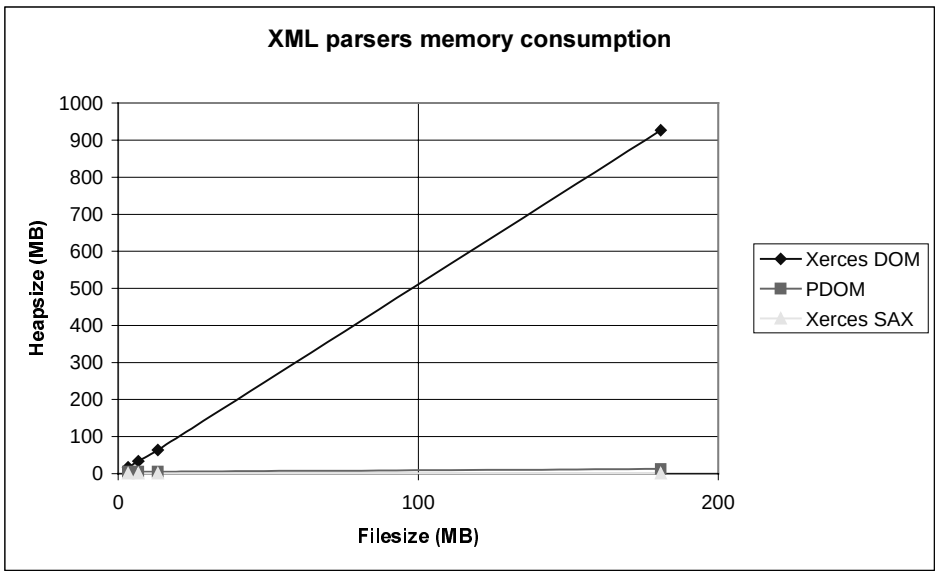

Figure 11: Parser memory consumption diagram

The diagram indicates that about 1000MB primary heap memory should be required to handle configuration files containing 50 000 MO's. This proves that the current implementation is insufficient when run on a system with a standard configuration.

4.6 Runtime Performance Comparison

The runtime-ratio between the three parsers varies a little depending on the size on the XML-file, which parser-specific option that are set, the speed of the hard-disk and a few other things. An average ratio is presented in Table 2.

Parser type	Runtime index
Xerces SAX-parser	1
Xerces DOM-parser	2
PDOM-parser	6

Table 2: Parser runtime comparison

This means e.g. that the SAX-parser is about six times faster than the PDOM-parser.

4.7 Parser Techniques Pros and Cons Summary

This section contains a summary of the pros and cons of the different parsing techniques.

4.7.1 DOM (Xerces)

This is the currently used parser.

+ No alteration of current code necessary
+ Fast
+ "Random access" of the XML nodes
+ Easy to use and thus easy maintenance
+ Freeware

+ Source-code available.

– Linearly growth of memory consumption
– No configurable error handling
– No commercial support available.

4.7.2 SAX

The SAX “serial” parser technique.

+ Fast
+ Low memory consumption
+ Configurable error handling
+ Freeware
+ Source-code available.

– Lots of code must be developed to handle parser callbacks and to store data-elements that are needed later on
– No commercial support available
– Only useful for the import functionality.

4.7.3 SAX and DOM Combined

The technique where SAX is used as the main parser and DOM used for building smaller data-trees.

+ Low memory consumption
+ Faster than the current implementation
+ Some of the code in the import function can be reused
+ Configurable error handling
+ Freeware
+ Source-code available.

– Some code must be developed to handle parser callbacks and storage of data-elements
– No commercial support available
– Only useful for the import functionality.

4.7.4 PDOM

The technique where the DOM data-tree is stored in a binary file.

+ Moderate memory consumption
+ Acceptable runtime performance[3]
+ “Random access” of the XML nodes
+ Can easily be integrated in the current implementation code
+ Commercial support available

– Only one implemented product available
– Cost for company license
– Dependence on one single third party company for future support and maintenance

[3] The runtime performance is presumed to be acceptable if the parser is faster than the direct use of the create_MO and set_MO CORBA operations. However, since no runtime tests has been made to measure the CORBA operations performance a qualified assumption were made about the relative performance.

– No Swedish distributor/support.

4.8 Comparison Tables

When comparing different methods, the following factors are considered to be important (most important factor first):

- Correct functionality (including acceptable memory consumption)
- Runtime performance
- Simple (not time-consuming) implementation and maintenance
- Fault handling and robustness

The cost of a commercial third party license is not included in the list but is implicitly considered to be important.

As previously mentioned, the current implementation is insufficient to cope with the requirements when executed on a standard system. The alternatives have their pros and cons. In the tables below the different alternatives are compared by applying probability estimates values (Prob) on the different factors and weight them with an importance value (Imp). It is considered that code of the current implementation is available when the values are estimated, i.e. the implementation of the alternatives could be faster if some code can be reused. The factor is graded from 0-10 where 10 are the best. The Prod value is calculated by multiplying Imp- and Prob-factors.

The currently used DOM-technique is not included in the tables because this evaluation presumes that is has to be replaced with another technique.

4.8.1 Import Comparison

		SAX		SAX/DOM combined		PDOM	
Description	**Imp**	**Prob**	**Prod**	**Prob**	**Prod**	**Prob**	**Prod**
Correct functionality	4	7	28	8	32	9	36
Runtime performance	3	10	30	9	27	4	12
Simple implementation	2	2	4	6	12	8	16
Fault handling and robustness	1	5	5	5	5	5	5
Low license cost	1	10	10	10	10	5	5
Summary		**77**		**86**		**74**	

Table 3: Import technique evaluation calculation

4.8.2 Export Comparison

		CS "hardcoded"		PDOM	
Description	**Imp**	**Prob**	**Prod**	**Prob**	**Prod**
Correct functionality	4	8	32	8	32
Runtime performance	3	10	30	4	12
Simple implementation	2	2	4	8	16
Fault handling and robustness	1	5	5	5	5
Low license cost	1	10	10	5	5
Summary		**81**		**70**	

Table 4: Export technique evaluation calculation

4.9 Summary and Recommended Solutions

As seen in Table 3 and Table 4 a combined SAX/DOM technique is recommended as the best choice for import and a CS “hardcoded” implementation the best for export. However, the differences are not significant and if some of the project conditions are changed, one of the other solutions could become the overall best.

5 Migrating Data Between Different MIM-versions

This chapter contains information about the second part of the thesis: how to make the migration between two different MIM-versions fully- or semi-automatic.

5.1 Problem Description

The different MO-types that are possible to store in the database are defined in a separate XML-file. This is a definition of the MIM. The MO-types in the XML-structure are converted to the different database object classes by means of a Model Parser component. When any change in the MIM is needed, e.g. a new MO-type or alteration of an existing MO-type or attribute, there is a high probability that the MIB that was formed according to the old MIM will be inconsistent. Today there is no standardised way to make the MIB consistent. This process could therefore be very time-consuming, especially when the MIB contains many MO-objects, and there is a need to make the process fully- or semi-automatic.

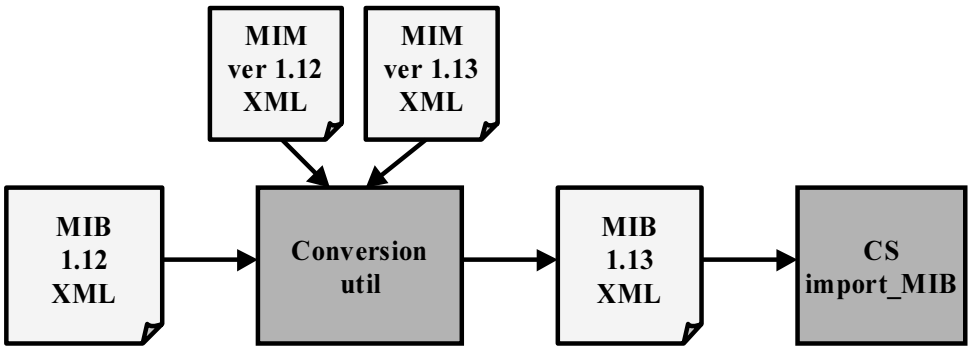

Figure 12: MIM/MIB migration overview

The conversion could either be done with an external tool or within the CS import method. Figure 12 shows the basic concept for an external conversion utility.

5.2 Conversion Action Analysis

This section lists all the possible conversion-actions that will be performed. For each type of action there is a definition of when it will occur.

Action	When
Fatal error	The MIB-def and the old MIM-def do not have the same version.
Delete MO	MO-type removed.
	The parent MO removed.
	New mandatory attribute has been added to the MO-type, or an attribute has been changed to mandatory.
Delete MO-attribute	The MO-attribute definition has been removed from MO-type.
	The data-type of the MO-attribute definition has been changed to an incompatible type.
	The MO-attribute definition contains an enum-type that has been removed or changed.
	The MO-attribute definition contains a struct or sequence that has been removed or changed.
Change attribute data-type	The data-type of the attribute has been changed to a compatible type, i.e. short -> long -> longlong -> float -> double.
Output an “Illegal value” log, but still transfer the MO-attribute	The range of the data-type in the MO-attribute definition has been changed and the MO-attribute value is “out of range”.
Delete relation	Relation-type has been removed.
	Relation includes a MO that has been deleted.

Table 5: Conversion actions analysis

With the current structure of the MO-type definitions in the MIM-file there are some type of changes that is impossible to distinguish. An example of this is if a MO-type has been renamed or divided into several different MO’s. Then there is no way to determine if a new MO-type has been added or the MO-type has been renamed. To be able to make this distinction, the format of the MO-type definitions must be changed so it is possible to include a reference to the old MO-type name.

5.3 Technique Analysis

There are several different techniques and tools that could be used to compare and convert XML-files. One of the tasks of the thesis was to find out if XSL could be a suitable technique. Other techniques that were considered interesting enough to evaluate were specialised 3PPs and the development of a new Java-application.

5.4 XSL

XSL is mainly a technique intended to convert one XML-document at a time, not to compare several documents. The significant pros and cons of XSL are listed below.

+ Advanced element filter/lookup functionality
– Large memory consumption
– Poor handling of multiple documents
– Poor debugging facilities
– Non intuitive syntax
– Limited variable handling

– No data-tree compare function

The large memory consumption is mainly due to the design of the XSL processor application. They are usually implemented in such way that they build at least three DOM-trees during the transformation: one for each source document, one for the target document and one for the XSL conversion definition. This characteristic makes XSL unsuitable to handle large MIB XML-files and should not be considered as the main tool in this context. However, XSL could be useful to compare the MIM-versions documents since they have moderate size.

5.5 3PP Application

One of the central parts of the problem is to track the changes between two different XML-files. This problem should be rather common in different XML-related applications, and therefore some time was spent to search for a 3PP product that implemented this function. However, the result of this search was modest. Only two applications were found and none of them were found to be of any significant use within this concept.

The two applications found were:

XMLTreeDiff (IBM)

According to the documentation, this application should be able to mark which nodes that have been changed. It requires an old version (1.1) of the xml4j-parser that was not possible to obtain. A test was made with version 1.1.4, but with erroneous result (method missing in Child-class).

Xmldiff (IBM)

Generates either graphical presentation of the differences or an XML file “tagged” with the differences. The tags are included as comments, which makes it hard to parse.

5.6 Java Application

A special Java-application can provide the most flexible solution to the migration problem. However, then all algorithms have to be implemented from scratch, which requires time-recourses.

The migration tool must be designed to handle large MIB XML-files. Therefore the main parser of the migration tool must be based on the SAX-technique. In the part that handles the comparison of the different MIM-versions, the DOM-technique could be used since the MIM-documents are smaller.

A prototype application was developed to prove the basic concepts of the migration.

5.6.1 Prototype Design

To have a flexible and open design the abstract Java-interface MIMChanges has been defined with methods that returns information about the changes of the different MIM-objects.

ConvertMIB is the main class that uses a SAX-parser to read the MIB XML-file. It queries the MIMChanges to find out which elements in the MIB-file that should be included in the new MIB-file.

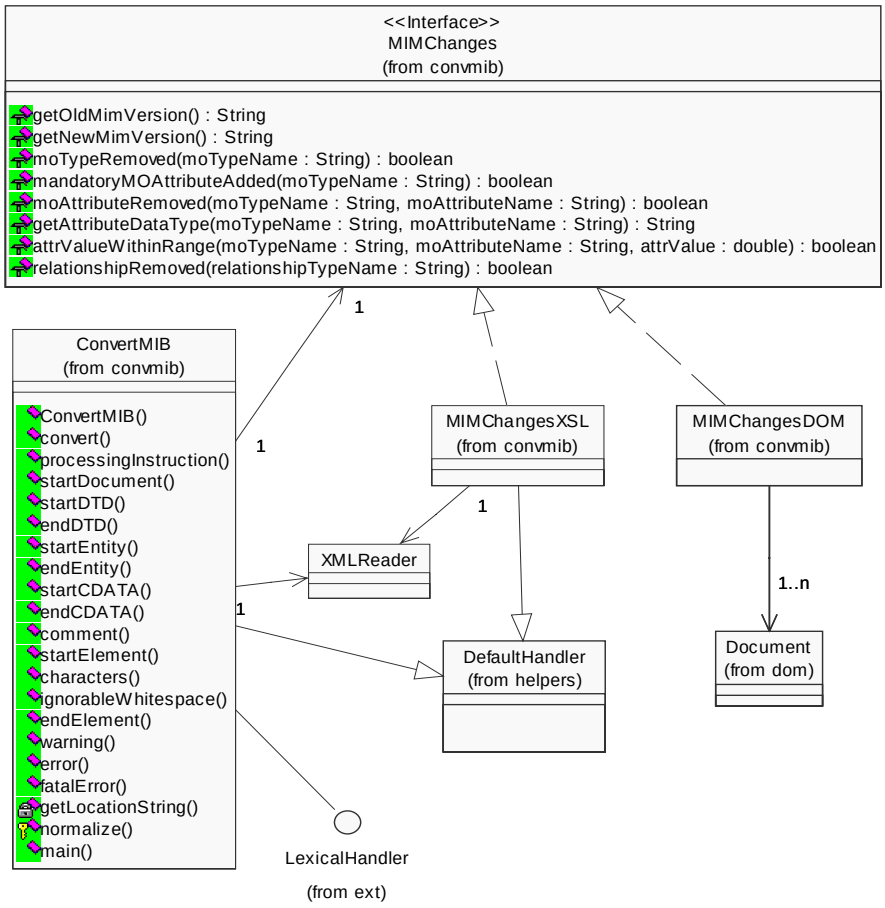

Figure 13: ConvertMIB design

Two different implementation of the interface has been done: MIMChangesXSL and MIMChangesDOM.

5.6.1.1 MIMChangesXSL

In the MIMChangesXSL implementation, XSL is used to extract the changes between the two MIM-definition versions. A special XSL-program, MIMDiff, has been implemented (see Appendix B:MIMDiff.xsl Source Code). MIMDiff stores the changes in a new XML-file with a special defined format. This file is then parsed by MIMChangesXSL, which builds an internal table of the changes.

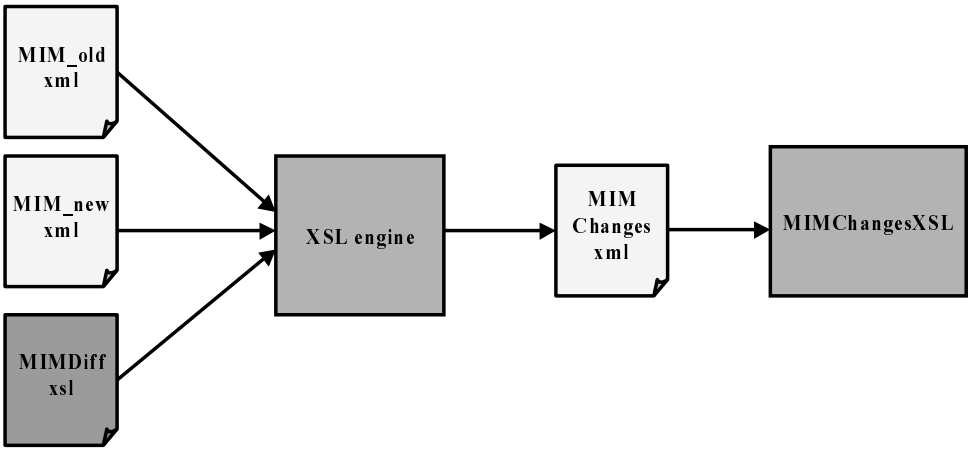

Figure 14: MIMChangesXSL concept

This solution requires an XSL processing engine to be installed in the system.

5.6.1.2 MIMChangesDOM

In the MIMChangesDOM implementation, the data from the two MIM-definition versions are received by means of the DOM-technique. The XML-files are parsed into two DOM-trees and then MIMChangesDOM queries the data-trees to extract the changes.

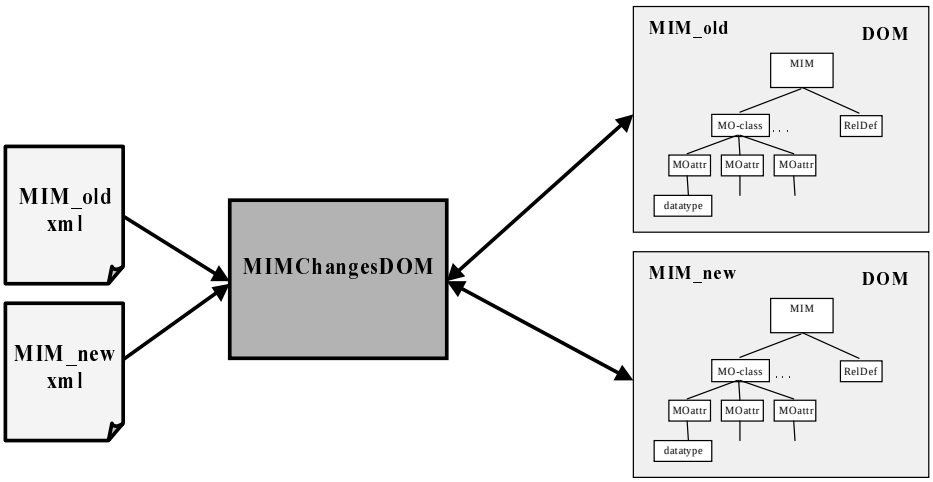

Figure 15: MIMChangesDOM concept

This implementation is more flexible and makes it possible to find differences at a more detailed level compared to the MIMChangesXSL.

5.7 Migration Within CS

Another way to handle the migration between different versions would be to make the conversion directly in the CS import_MIB component. This section discusses two such concepts.

5.7.1 LDAP Query

All versions of the MIM-definitions are stored in a directory service that is accessible from the import_MIB component. This makes it possible to do an implementation of the MIMChanges-interface where the MIM-data is read from the directory service (LDAP).

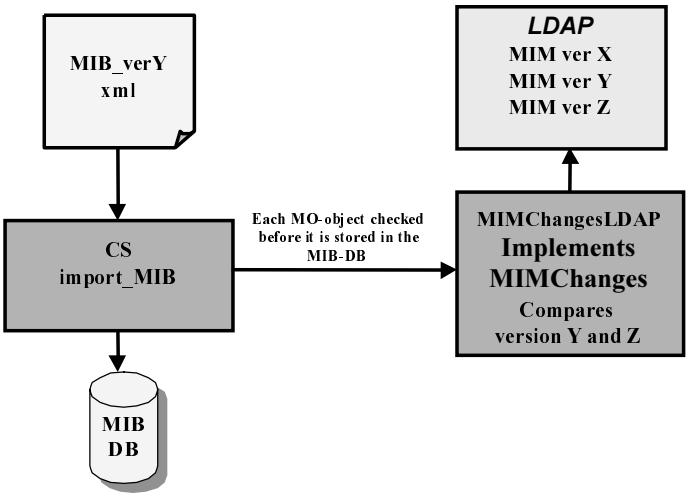

Figure 16: MIMChangesLDAP concept

When an older MIB-version is imported, a new MIMChangesLDAP object is created that reports the differences between the current MIM-version and the MIM-version the MIB-file is based on.

The import_MIB check each MIB-element before it is stored in the database.

The main pros and cons of this design are:

\+ The user does not have to run an external conversion utility where the different MIM XML-files must be supplied.

– The user has less control over the conversion process
– An export from the database has to be done to produce a MIB XML-file that comply to the current MIM-format
– Any problem in this function will affect the stability and robustness of the whole CS-component.

5.7.2 MO-conversion Plug-in

Another conversion concept would be to make a “hard-coded” conversion code-module for each new MIM-version. There must then be a conversion-handler defined for each MO-type in each previous MIM-version. The conversion plug-in modules could with advantage be defined by the person who makes the MIM-definition changes.

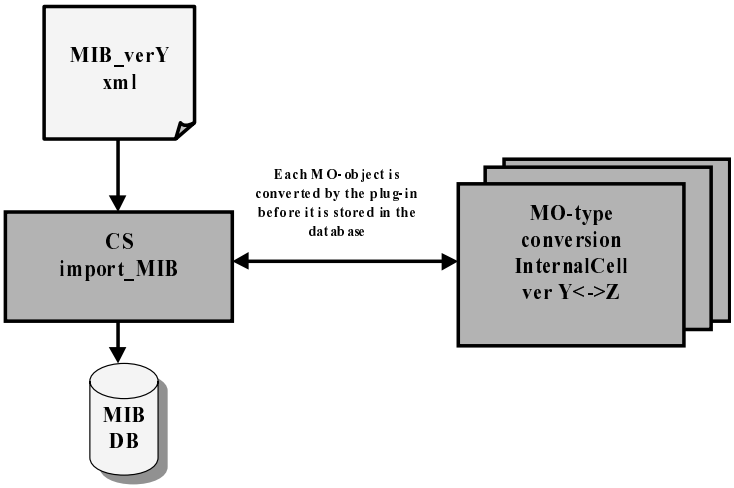

Figure 17: Plug-in MIB-conversion concept

The main pros and cons of this design are:

+ Possible to have special conversion-rules for each MO-type. This could, for example, make it possible to handle renaming of MO-types.

– A lot of code has to be implemented each time a new MIM-version is installed
– The user has less control over the conversion process
– An export from the database has to be done to produce a MIB XML-file that comply to the current MIM-format.

5.8 Summary and Recommended Solutions

A migration tool will with high probability be cost effective to implement.

The XSL-technique is not so useful within this area and an implementation in Java will be more effective.

It is difficult to suggest just one of the designs as the best. If the migration should be done frequently then an implementation within the import_MIB module would probably be most effective, otherwise an external tool should be considered.

6 DTD/Schema Format

This chapter contains information about the third part of the thesis: an evaluation of which DTD/schema format that is most effective to use for the MIB XML-file.

6.1 Problem Description

The current format of the MIB XML-file is specified by the DTD-definition in the mib.dtd file (see Appendix C: DTD/schema Formats). There is a need to make the MIB XML-format as effective as possible with respect to the following factors: correct functionality, easy implementation, simple readability and good runtime performance.

There are several “standard” XML-formats available that are intended to be used to define/exchange different kinds of object oriented data. The main task of this part of the thesis is to analyse and compare those formats in respect to the current mib.dtd format. The formats that were suggested to be analysed are XMI, CCM IRP and VXML. There should also be investigated if the mib.dtd format could be improved in any way.

6.2 CCM IRP

Common Configuration Management Integration Reference Points (CCM IRP) is a format defined by the 3GPP-organisation. It is intended to facilitate the exchange of object configuration data between different parts in the third generation mobile network, regardless of the vendor.

The main pros and cons of CCM IRP are:

+ “Standard” format -- the MIB-files can be reused in other applications without modification
+ Each MO contains Distinguished Name -- simplifies the import parsing.

– MOs are needed to be stored in the correct order in the file
– Uses one XML-element type for all primitive data-types -- extra type-handling code necessary
– Not possible to define associations or hoppers.

The most significant disadvantage is the lack of association- and hopper-definition. Since Ericsson is a member of the 3GPP-organisation it should be possible get rid of this limitation by suggesting an extension of the format.

6.3 VXML

Versant XML (VXML) is used to define data that should be imported directly in a Versant Object Oriented database by means of their import utility.

Figure 18: VXML concept

The main pros and cons of VXML are:

+ Simple format -- easy to understand
+ Fast import -- the database is queried directly.

– The structure of the XML-import file must follow the class-structure in the database -- any changes in the database-structure affects the format of the MIB XML-file
– Versant's import tool has limited constraint checks -- no warnings will be generated if MO-attributes etc are missing or out of range.

The absence of constraint checks in the VXML import utility makes this import technique insufficient in respect of the CS-requirements.

If the import would be done through the import_MIB, or some other external application with a constraint check included, the VXML DTD-format could theoretically be used. However, then there will be no advantage over using the current mib.dtd format and there will be similar drawbacks as for the CCM IRP format.

6.4 XMI

XML Metadata Interchange Format (XMI) is a format defined by the Object Management Group.

The main purpose of XMI is to enable easy interchange of metadata between modelling tools (based on the OMG UML) and between tools and metadata repositories (OMG MOF based) in distributed heterogeneous environments. (OMG 2000)

XMI is a complex standard. The specification document (OMG 2000) covers 400 pages. The specification does not specify a fixed DTD-format; it only specifies the basic guidelines for constructing a valid XMI DTD.

The conclusion of the analyse is that the XMI-technique is not so useful within the import_MIB/CS-context.

6.5 Mib.dtd

The DTD-definition in the mib.dtd file defines the current format of the MIB XML-file.

The main pros and cons of mib.dtd are:

+ Currently used format -- import_MIB component and the specialised MIB generation tools do not have to be changed
+ Special XML-tags for each available primitive type of the MO-attributes -- simplifies the consistency check.

- The defined MO-instances do not have distinguished name. The hierarchy in the XML-file defines the MO-hierarchy. The current element-level within the XML-file has to be tracked
- Includes XML-elements with no significant functionality -- increases memory consumption and parsing time.

The change of a format that is already in use in delivered systems has to have a strong motivation; i.e. the system performance must be significantly increased. When mib.dtd are compared to the other formats, no really significant fact can be found that motivates a switch to a new format.

6.5.1 Improvements

Another question was if the mib.dtd format itself could be changed in such a way that performance increases, but with little impact on the import_MIB component and the specialised MIB XML-file generation tools?

The easiest way to improve parsing performance would be to reduce the number of XML-elements.

Each MO-attribute definition has a datatype and value-element. Those elements have no significant function when the MIB XML-files are imported.

If the mib.dtd file is changed according to the suggestion in Appendix C: DTD/schema Formats, the DOM memory consumption and parsing time could be reduced by 20-40%. Only minor changes in the import_MIB component should be needed to handle the changed format. Other changes could improve the memory and XML-parsing performance even more, but the performance-loss in other parts of the import_MIB will then level out those improvements. The suggested changes are estimated to be the best compromise when all parts of the import procedure are considered

6.6 Summary and Recommended Solutions

CCM IRP could be useful with some changes implemented.

VXML and XMI are of less interest. VXML will make the constraint check more difficult and is also too associated to the database-structure. XMI is very complex and has no fixed dtd-definition and will therefore not be efficient.

Mib.dtd is recommended to be used within the foreseeable future, but some improvements should be considered.

7 Summary and Future Work

This chapter contains a summary of the thesis. It also contains suggestions about how the results should be used and recommendations for future work.

7.1 Large Import and Export Files

The current implementation of the import/export function is based on an in-memory DOM-parser, which has a heap-memory consumption that is linear to the size of the XML-file. This will cause a primary memory shortage problem when a network with a large number of elements should be configured/simulated. To solve this problem, the current methods must be based on a different technique.

The techniques evaluated were the SAX-parser, DOM-parser with alternative storage technique and a “hard-coded” solution.

The SAX-parser is fast with low memory consumption, but is usually complex to use and requires more code at the application side. The DOM-parser technique is easy to use and integrate in the current system, but it consumes a lot of memory for the storage of the data-tree that is built. A “hard-coded” solution makes it possible to have full control of the memory consumption and other parts of the process, but it will require a lot of coding.

When comparing the different techniques the following factors were considered: correct functionality, runtime performance, simple (not time-consuming) implementation and maintenance, fault handling/robustness and cost for software license.

7.1.1 Conclusions

A combined SAX/DOM technique is recommended as the best choice for import and a CS “hardcoded” implementation the best for export. However, the differences are not significant and if some of the project conditions are changed, one of the other solutions could become the overall best.

7.2 Migrating Data Between Different MIM-versions

There is a need to make the migration between different network-models fully- or semi-automatic. In this part of the thesis different techniques for comparison and conversion of XML-files were evaluated to find a solution to the problem.

The techniques evaluated were XSL, 3PP and a Java-based solution.

XSL has advanced element filter/lookup functionality, but the XSL-engine consumes a lot of memory and its ability to compare several XML-documents is limited. Two 3PP were evaluated, but none of them was found suitable within this area. The Java-based solution can facilitate the most flexible conversion utility, but it will require a lot of code-implementation. The conversion can be implemented as a separate external tool, or as part of the import function in the CS.

7.2.1 Conclusions

A migration tool will with high probability be cost effective to implement.

The XSL-technique is not so useful within this area and an implementation in Java will be more effective.

It is difficult to suggest just one of the designs as the best. If the migration should be done frequently then an implementation within the import_MIB module would probably be most effective, otherwise an external tool should be considered.

7.3 DTD/Schema Format

In the third part of the thesis the format of the XML-file was studied to see if any changes could increase the performance.

The following formats were analysed: XMI, CCM IRP and VXML. The current mib.dtd format was also analysed to find out if it could be improved in any way.

The main purpose of XMI is to enable easy interchange of metadata between modelling tools. The specification does not specify a fixed DTD-format; it only specifies the basic guidelines for constructing a valid XMI DTD.

CCM IRP is intended to facilitate the exchange of object configuration data between different parts in the third generation mobile network, regardless of the vendor. It is a “standard” format, which makes it possible to reuse the MIB-files in other applications without modification. Each MO contains Distinguished Name, which simplifies the import parsing. However, it uses only one XML-element type for all primitive data-types, which requires extra type-handling code. Furthermore, it does not support definition of associations or hoppers, which is another drawback.

Versant XML (VXML) is used to define data that should be imported directly in a Versant Object Oriented database by means of their import utility. The import is very fast if Versants VXML-tool is used because the database is queried directly. However, any changes in the database-structure will affects the format of the MIB XML-file and the constraint checks are very limited.

The DTD-definition in the mib.dtd file defines the current format of the MIB XML-file. The format could be improved by removing elements with no significant functionality. This will reduce the memory consumption and parsing time.

7.3.1 Conclusions

CCM IRP could be useful with some changes implemented.

VXML and XMI are of less interest. VXML will make the constraint check more difficult and is also too associated to the database-structure. XMI’s drawbacks of a complex characteristic and no fixed dtd-definition makes it unsuitable.

Mib.dtd is recommended to be used within the foreseeable future, but some improvements should be considered.

7.4 Future Work

This thesis has proven the basic concepts by means of prototype programs. The next step would be to implement the suggested solutions in the current CS-system.

The performance issues could be further investigated by analyzing the use of the CORBA methods create_mo and set_mo methods compared to the import method. The transaction handling to the database could probably be improved by means of some optimisation algorithm. Other areas that could be further investigated are the fault handling and robustness.

The XML is a new technique that is rapidly spreading. New tools are constantly produced and there is a high probability that some of them will be useful within the areas covered in the thesis. There are, for example, new validating techniques like XML Schema that makes it possible to let the XML-parser do some of the constraint checks that is performed within CS today. It is therefore a good idea to continuously follow the development in the XML-area.

8 Acronyms and Abbreviations

3GPP	Third Generation Partnership Project
3PP	Third Party Product
API	Application Programming Interface
BSC	Base Station Controller
BSS	Base Station System
CCM IRP	Common Configuration Management IRP
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CS	Configuration Service
DN	Distinguished Name
DOM	Document Object Model
DTD	Document Type Definition
GSM	Global System for Mobile communication
HTML	HyperText Markup Language
IIOP	Internet Inter-ORB Protocol
IRP	Integration Reference Point
ISO	International Organisation for Standardisation
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MIB	Managed Information Base
MIM	Managed Information Model
MO	Managed Object
MOF	Meta Object Facility
MSC	Mobile Services switching Center
OMG	Object Management Group
OS	Operating System
OSS	Operation and Support System
PDF	Portable Document Format
PDOM	Persistent DOM
PLMN	Public Land Mobile Network
RANOS	Radio Access Network Operation System
SAX	Simple API for XML
SS	Supplementary Services
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
UTF	Unicode Transformation Format
VXML	Versant XML

XMI	XML Metadata Interchange
XML	eXtended Markup Language
XPath	XML Path language
XQL	XML Query Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
W3C	World Wide Web Consortium

9 References

Birbeck, Mark et.al. (2000), *Professional XML*, Birmingham, Wrox, 1st edition, 1-86100-311-0

CS (2000), *Programmer's Guide – Configuration Service* (198 17-APR 90 161 Uen Rev PF4 2000-10-04), Ericsson Radio Systems

GMD (2001), GMD-IPSI XQL Engine, http://xml.darmstadt.gmd.de/xql/

KMS (2001), Kings Mountain Systems, http://www.kingsmountain.com/ldapRoadmap.shtml

Morrison, Michael (2000), *XML unleashed*, Indianapolis (Ind.), Sams, 0672315149

OMG (2000), OMG XML *Metadata Interchange (XMI) Specification 1.1*, OMG, ftp://ftp.omg.org/pub/docs/formal/00-11-02.pdf

OMG (2001), Object Management Group, http://www.omg.org/

Sun (2001), Sun's source for Java technology, http://java.sun.com/

W3C (2001), World Wide Web Consortium, http://www.w3.org/

Appendix A: PDOM

This appendix contains information about the GMD-IPSI XQL Engine. Most of the information has been fetched from their web-site located at: http://xml.darmstadt.gmd.de/xql/.

GMD Persistent DOM

The PDOM class allows generating binary, indexed files containing a persistent W3C-DOM. A PDOM file immediately offers all DOM operations without the cost of parsing XML or building an in-memory DOM representation first. Combined with servlets and XQL, PDOM files offer an efficient method to serve XML fragments from large documents. A PDOM file may be created from any XML file or programmatically using W3C-DOM methods.

When creating PDOM files from XML files, SAX events are used to communicate with the XML parser. Using the event based SAX API there never has to be a full representation of your XML file in main memory. Because of this the size of a PDOM file is only limited by disk space, not by main memory.

The de.gmd.ipsi.pdom.PDocument class implements org.w3c.dom.Document, so the PDOM may be used anywhere a W3C compliant DOM implementation is needed. As the PDOM API supports all methods of the W3C-DOM, including updates and inserts, programmatic creation and modification of PDOM files is possible.

Overview of the PDOM Features

This section describes the different special characteristic of the PDOM

Caching: A PDOM file is organized in pages, each containing 128 DOM nodes of variable length. When a PDOM Node is accessed by a W3C-DOM method, the containing page is loaded into a main memory cache. Starting with a default cache size of 100 pages (12.800 DOM Nodes), the main memory cache can be resized any time. It will, however, never shrink below 20 pages (2.560 DOM Nodes). It is recommended to use the largest cache size your machine's main memory can hold without swapping, as a larger cache improves overall PDOM performance. The same cache is shared by all PDOM documents opened with the same instance of the PDOM engine. The caching strategy used is "least recently used" (LRU).

Defragmentation: When a node is programmatically inserted, updated or delete by W3C-DOM methods, the page containing the node is invalidated ("dirty page"). If a dirty page is displaced from the cache, the modified page is appended at the end of the PDOM file. So a PDOM file will grow during write operations, as the file space occupied by invalidated pages will not be removed or reused automatically. Note that just reading and or querying a PDOM file, however, will never change the file size.

The PDOM file can be defragmented at any time by removing unused pages. During this operation a temporary file containing only valid pages is created and finally the fragmented PDOM file is replaced with the unfragmented copy. It is possible to define the directory where the temporary file is created. The slack ratio, that is the percentage of wasted file space divided by physical file size can be accessed by user applications. The number is normalized to a double between 0.0 and 1.0. It is up to the user application to start a defragmentation, probably if the slack ratio grows beyond a tolerable mark.

Full garbage collection: Defragmentation does work on a per-page basis and does not free space occupied by DOM nodes that have been deleted within pages. To also free this space, a full garbage collection is required. To avoid dangling object references, a garbage collection is only safe if the PDOM file is not opened by another PDOM engine and no PDocument object is currently bound to the PDOM file. This also includes any child nodes of PDocument, which may still be in main memory left from previous operations. It is the duty of the user application to enforce these conditions; else you are in danger to garble the PDOM file. Full garbage collection includes defragmentation.

Commit points: At any time a user application doing update, delete or insert operations on a PDOM can decide to commit the current status quo of the PDOM. In the commit operation the main file index, normally maintained in main memory, is written back to disk. If the user application crashes, e.g. because of a "disk full" error, the PDOM will be in the state it was immediately before the last successful commit operation when re-opened. Great care was taken to ensure file consistency even after crashes. There is, however, a minimal chance of corrupting a file if the user application dies during a commit operation. Keep in mind that the PDOM does not try to be a fully-fledged database.

Compression with gzip: Optionally a PDOM file can be compressed on the fly using the gzip algorithm. This will result in smaller files, usually half the size of an uncompressed PDOM file. The trade-off here is speed: a compressed PDOM file usually increases the execution for reading and writing pages by 20%. Compression is a one-time decision take at creating time of the PDOM file. A file can not be compressed later. All operations opening PDOM files will automatically recognize compression and handle this fact transparently. User applications never have to care or know about compression when dealing with existing PDOM files.

Multithreaded access: The same PDOM file can be read by multiple threads in parallel without problem. Update operations block read and write operations for other threads. Given this, all atomic operations on a PDOM file are thread safe. However, composed update operations (e.g. reading a node, modifying it and write back to the PDOM) suffer from the well-known transaction difficulties. To ensure atomicity of complex updates, the application has to synchronize the critical block of code with the PDocument object.

Installation

To integrate the GMD-IPSI XQL with your Java based XML environment simply download the distribution after agreeing to the license terms,

- add the JAR file contained in the distribution to your Java CLASSPATH,
- add a SAX parser to your CLASSPATH (if not already there)

- add a W3C-DOM implementation to your CLASSPATH (if not already there)

The GMD-IPSI PDOM engine requires a third party SAX parser to read XML documents. The SAX API does not provide full information on XML documents, e.g. comments or CDATA sections are missing. As a result, such nodes are missing from the PDOM when built using SAX events. Glue code for IBM's xml4j2 parser is included, using its proprietary extensions to SAX, to create DOM nodes of types not supported by standard SAX. So xml4j2 is not required, but recommended to be used as SAX parser. To automatically use this feature, simply add xml4j2 to your CLASSPATH.

Any other SAX compliant parser may be specified in the command line tools. If no parser is specified and xml4j2 is missing, XP, Microstar Ælfred, Sun Project X and Oracle XML parser are auto-detected and used if present in your CLASSPATH.

If you want to query only PDOM files, no third party W3C-DOM implementation is needed. When the XQL command line tool is used to query arbitrary XML documents, a temporary in-memory DOM is built. Again you may explicitly specify the W3C-DOM implementation you want to use. If no preferred DOM implementation is given, Open XML DOM, Sun Project X, xml4j, xml4j2, Docuverse DOM SDK and Oracle XML DOM are auto-detected and used if present in your CLASSPATH.

A new package with DOM utility functions has been added. To use the included methods to instantiate a W3C-DOM from HTML or Microsoft RTF, the Sun JFC library (aka Swing 1.1) has to be in your CLASSPATH.

Development is done using Sun JDK 1.2.1, the jikes Java compiler 0.47 and xml4j 2.0.9. The engine is also tested with the Sun JDK 1.1.8, Microsoft SDK 3.2 and the parsers and DOM implementations listed above on an irregular basis.

Creating a PDOM File

There are two ways to create a PDOM file, either by writing an in-memory DOM to disk or by creating it from an XML InputStream.

Variant 1 demonstrates the creation of a PDOM file from an in-memory instance of another DOM. Any W3C-DOM implementation can be used. The example does use the gzip compression option to create a compressed PDOM file.

Variant 2 demonstrates the creation of a PDOM file from a vanilla plain XML file. The built-in validating SAX parser, extending xml4j2's com.ibm.xml.parsers.SAXParser, is used. As we decide to use validation, it is feasible to suppress ignorable whitespace. This way a lot of unnecessary Text nodes holding only whitespace are suppressed, resulting in a smaller, faster PDOM file.

```
import de.gmd.ipsi.pdom.*;
import de.gmd.ipsi.domutil.*;
import org.w3c.dom.Document;
```

```
//
// Variant 1: Writing an in-memory DOM Document to disk
//

// A Document created by your favorite DOM implementation
Document in_memory_doc = DOMUtil.createDocument();
PDOM.writeDOMFile(
     "mydoc.pdom",
     in_memory_doc,
     true // false = no gzip compression, true = create
gziped PDOM
);

//
// Variant 2: Create a PDOM by parsing an XML input stream
//

Document pdoc = new PDocument("mydoc.pdom");
DOMUtil.parseXML(
     new FileInputStream("valid_with_dtd.xml"),
     pdoc,  // The Document's factory is used to create
PDOM Nodes
     true,  // Parse mode: true = validating, false = non-
validating
     DOMUtil.SKIP_IGNORABLE_WHITESPACE // Whitespace
treatment, see API docs
);
((PDocument)pdoc).commit();       // be sure to flush to
disk
```

Licence

A commercial version is available with the possibility to buy a company-license.

Licensing and pricing for commercial use:

The Base Package of the Infonyte XQL Suite contains the XQL query engine, the PDOM storage component, command line front-ends, and user documentation. The following license types are available:

Single User Runtime License
Includes: Infonyte XQL Suite Base Package
EURO 100

Single Developer License
Includes: Infonyte XQL Suite Base Package, API docs
EURO 1000

Server Runtime License (per CPU)
Includes: Infonyte XQL Suite Base Package, API docs, Servlet
EURO 1500

Company License
Proposals available on request
OEM Runtime Licenses

Proposals available on request

More information can be found at: http://www.globit.com/infonyte.htm and http://xml.darmstadt.gmd.de/xql/.

Appendix B: MIMDiff.xsl Source Code

The MIMDiff prototype program were implemented in Java and XSL. This appendix contains the source code of the XSL part. The purpose of this appendix is to present a practical example of an XSL-implementation in general and the use of XSL for comparison of two XML-files in particular.

XSL source code

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version = "1.0">

<xsl:output method = "xml" omit-xml-declaration = "no" indent="yes" />

<xsl:variable name="old" select="document('bsc_mim_old.xml')" />
<xsl:variable name="new" select="/" />


<xsl:template match="/">
<xsl:element name="mimdiff">
 <xsl:attribute name="oldver">
  <xsl:value-of select="$old/models/mim/@version" />
 </xsl:attribute>
 <xsl:attribute name="oldrelease">
  <xsl:value-of select="$old/models/mim/@release" />
 </xsl:attribute>
 <xsl:attribute name="newver">
  <xsl:value-of select="$new/models/mim/@version" />
 </xsl:attribute>
 <xsl:attribute name="newrelease">
  <xsl:value-of select="$new/models/mim/@release" />
 </xsl:attribute>

<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">struct_added</xsl:with-param>
  <xsl:with-param name="more" select="$new/models/mim/struct" />
  <xsl:with-param name="less" select="$old/models/mim/struct" />
</xsl:call-template>

<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">struct_removed</xsl:with-param>
  <xsl:with-param name="more" select="$old/models/mim/struct" />
  <xsl:with-param name="less" select="$new/models/mim/struct" />
</xsl:call-template>

<xsl:call-template name="list-altered">
  <xsl:with-param name="more" select="$new/models/mim/struct" />
  <xsl:with-param name="less" select="$old/models/mim/struct" />
</xsl:call-template>


<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">enum_added</xsl:with-param>
  <xsl:with-param name="more" select="$new/models/mim/enum" />
  <xsl:with-param name="less" select="$old/models/mim/enum" />
</xsl:call-template>

<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">enum_removed</xsl:with-param>
  <xsl:with-param name="more" select="$old/models/mim/enum" />
  <xsl:with-param name="less" select="$new/models/mim/enum" />
</xsl:call-template>

<xsl:call-template name="list-altered">
  <xsl:with-param name="more" select="$new/models/mim/enum" />
  <xsl:with-param name="less" select="$old/models/mim/enum" />
```

```
</xsl:call-template>


<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">class_added</xsl:with-param>
  <xsl:with-param name="more" select="$new/models/mim/class" />
  <xsl:with-param name="less" select="$old/models/mim/class" />
</xsl:call-template>

<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">class_removed</xsl:with-param>
  <xsl:with-param name="more" select="$old/models/mim/class" />
  <xsl:with-param name="less" select="$new/models/mim/class" />
</xsl:call-template>

<xsl:call-template name="list-altered">
  <xsl:with-param name="element-name">class_altered</xsl:with-param>
  <xsl:with-param name="more" select="$new/models/mim/class" />
  <xsl:with-param name="less" select="$old/models/mim/class" />
</xsl:call-template>


<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">relationship_added</xsl:with-param>
  <xsl:with-param name="more" select="$new/models/mim/relationship" />
  <xsl:with-param name="less" select="$old/models/mim/relationship" />
</xsl:call-template>

<xsl:call-template name="list-additional">
  <xsl:with-param name="element-name">relationship_removed</xsl:with-param>
  <xsl:with-param name="more" select="$old/models/mim/relationship" />
  <xsl:with-param name="less" select="$new/models/mim/relationship" />
</xsl:call-template>

<xsl:call-template name="list-altered">
  <xsl:with-param name="more" select="$new/models/mim/relationship" />
  <xsl:with-param name="less" select="$old/models/mim/relationship" />
</xsl:call-template>

</xsl:element>
</xsl:template>


<xsl:template name="list-additional">
  <xsl:param name="element-name" />
  <xsl:param name="more" />
  <xsl:param name="less" />
  <!-- cycle through each item in the 'more' document -->
  <xsl:for-each select="$more">
    <xsl:variable name="item" select="." />
    <xsl:variable name="comp" select="$less[@name=$item/@name]"/>
      <xsl:if test="count($comp) = 0">
         <xsl:element name ="{$element-name}">
           <xsl:attribute name = "name" >
             <xsl:value-of select="$item/@name" />
           </xsl:attribute>
         </xsl:element>
      </xsl:if>
  </xsl:for-each>
</xsl:template>

<xsl:template name="list-additional-attributes">
  <xsl:param name="more" />
  <xsl:param name="less" />
  <!-- cycle through each item in the 'more' document -->
  <xsl:for-each select="$more">
    <xsl:variable name="item" select="." />
    <xsl:variable name="comp" select="$less[@name=$item/@name]"/>
```

```
      <xsl:if test="count($comp) = 0">
         <xsl:element name ="attribute_added">
           <xsl:attribute name = "name" >
             <xsl:value-of select="$item/@name" />
           </xsl:attribute>
           <xsl:attribute name = "mandatory" >
             <xsl:if test="count(./mandatory) > 0">
               <xsl:value-of select="'true'" />
             </xsl:if>
             <xsl:if test="count(./mandatory) = 0">
               <xsl:value-of select="'false'" />
             </xsl:if>
           </xsl:attribute>
         </xsl:element>
      </xsl:if>
  </xsl:for-each>
</xsl:template>


<xsl:template name="list-altered2">
  <xsl:param name="element-name" />
  <xsl:param name="more" />
  <xsl:param name="less" />
  <!-- cycle through each item in the 'more' document -->
  <xsl:for-each select="$more">
    <xsl:variable name="item" select="." />
      <xsl:for-each select="$less">
      <xsl:if test="$item/@name = ./@name">
        <xsl:call-template name="list-additional">
          <xsl:with-param name="element-name">attribute_removed</xsl:with-param>
          <xsl:with-param name="more" select="." />
          <xsl:with-param name="less" select="$item" />
        </xsl:call-template>
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>


<xsl:template name="list-altered">
  <xsl:param name="more" />
  <xsl:param name="less" />
  <!-- cycle through each item in the 'more' document -->
  <xsl:for-each select="$more">
    <xsl:variable name="item" select="." />
    <xsl:variable name="diffmore" select="." />
    <!-- select the 'less' document so that elements can be selected from that
         document -->
    <xsl:for-each select="$less">
      <xsl:variable name="olditem" select="." />
      <xsl:variable name="diffless" select="." />
      <xsl:if test="$item/@name = $olditem/@name">
           <xsl:element name ="{concat(name($item),'_changes')}">
             <xsl:attribute name = "name" >
               <xsl:value-of select="$item/@name" />
             </xsl:attribute>
               <xsl:choose >
                 <xsl:when test="name($item) = 'struct'">
                   <xsl:call-template name="list-additional">
                     <xsl:with-param name="element-name">structMember_added</xsl:with-
param>
                     <xsl:with-param name="more" select="$item/structMember" />
                     <xsl:with-param name="less" select="$olditem/structMember" />
                   </xsl:call-template>
                   <xsl:call-template name="list-additional">
                     <xsl:with-param name="element-
name">structMember_removed</xsl:with-param>
                     <xsl:with-param name="more" select="$olditem/structMember" />
```

```
                    <xsl:with-param name="less" select="$item/structMember" />
                  </xsl:call-template>
                  <xsl:call-template name="list-altered">
                    <xsl:with-param name="more" select="$item/structMember" />
                    <xsl:with-param name="less" select="$olditem/structMember" />
                  </xsl:call-template>
                </xsl:when>
                <xsl:when test="name($item) = 'enum'">
                  <xsl:call-template name="list-additional">
                    <xsl:with-param name="element-name">enumMember_added</xsl:with-
param>
                    <xsl:with-param name="more" select="$item/enumMember" />
                    <xsl:with-param name="less" select="$olditem/enumMember" />
                  </xsl:call-template>
                  <xsl:call-template name="list-additional">
                    <xsl:with-param name="element-name">enumMember_removed</xsl:with-
param>
                    <xsl:with-param name="more" select="$olditem/enumMember" />
                    <xsl:with-param name="less" select="$item/enumMember" />
                  </xsl:call-template>
                </xsl:when>
                <xsl:when test="name($item) = 'class'">
                  <xsl:call-template name="list-additional-attributes">
                    <xsl:with-param name="more" select="$item/attribute" />
                    <xsl:with-param name="less" select="$olditem/attribute" />
                  </xsl:call-template>
                  <xsl:call-template name="list-additional">
                    <xsl:with-param name="element-name">attribute_removed</xsl:with-
param>
                    <xsl:with-param name="more" select="$olditem/attribute" />
                    <xsl:with-param name="less" select="$item/attribute" />
                  </xsl:call-template>
                  <xsl:call-template name="list-altered">
                    <xsl:with-param name="more" select="$item/attribute" />
                    <xsl:with-param name="less" select="$olditem/attribute" />
                  </xsl:call-template>
                </xsl:when>
                <xsl:when test="name($item) = 'attribute'">
                  <xsl:call-template name="analyse-attribute-changes">
                    <xsl:with-param name="new" select="$item/." />
                    <xsl:with-param name="old" select="$olditem/." />
                  </xsl:call-template>
                </xsl:when>
                <xsl:when test="name($item) = 'structMember'">
                  <xsl:call-template name="analyse-dataTree-changes">
                    <xsl:with-param name="new" select="$item/." />
                    <xsl:with-param name="old" select="$olditem/." />
                  </xsl:call-template>
                </xsl:when>
              </xsl:choose>

          </xsl:element>
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>


<xsl:template name="analyse-dataTree-changes">
  <xsl:param name="new" />
  <xsl:param name="old" />
  <xsl:variable name="newNodeNamesStr" >
  <xsl:for-each select="$new//*">
    <xsl:value-of select="name(.)" />
  </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="oldNodeNamesStr" >
  <xsl:for-each select="$old//*">
```

```
    <xsl:value-of select="name(.)" />
  </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="newNodeValuesStr" >
  <xsl:for-each select="$new//*">
    <xsl:value-of select="." />
  </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="oldNodeValuesStr" >
  <xsl:for-each select="$old//*">
    <xsl:value-of select="." />
  </xsl:for-each>
  </xsl:variable>
  <xsl:if test="$newNodeNamesStr != $oldNodeNamesStr">
    <xsl:element name="dataElementNamesChanged" />
  </xsl:if>
  <xsl:if test="$newNodeValuesStr != $oldNodeValuesStr">
    <xsl:element name="dataValuesChanged" />
  </xsl:if>
</xsl:template>



<xsl:template name="analyse-attribute-changes">
  <xsl:param name="new" />
  <xsl:param name="old" />
  <xsl:variable name="newDataType" select="$new/dataType/*[position()=1]" />
  <xsl:variable name="oldDataType" select="$old/dataType/*[position()=1]" />
  <xsl:variable name="newDataTypeNodeStr" >
  <xsl:for-each select="$new/dataType//*">
    <xsl:value-of select="name(.)" />
    <xsl:value-of select="." />
  </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="oldDataTypeNodeStr" >
  <xsl:for-each select="$old/dataType//*">
    <xsl:value-of select="name(.)" />
    <xsl:value-of select="." />
  </xsl:for-each>
  </xsl:variable>
  <xsl:if test="count($new/mandatory) > count($old/mandatory)">
    <xsl:element name="mandatory" />
  </xsl:if>
  <xsl:element name ="newDataType">
    <xsl:attribute name = "type" >
      <xsl:value-of select="name($newDataType)" />
    </xsl:attribute>
    <xsl:if test="name($newDataType) = 'structRef' or name($newDataType) = 'enumRef'">
      <xsl:attribute name = "name" >
        <xsl:value-of select="$newDataType/@name" />
      </xsl:attribute>
    </xsl:if>
  </xsl:element>
  <xsl:element name ="oldDataType">
    <xsl:attribute name = "type" >
      <xsl:value-of select="name($oldDataType)" />
    </xsl:attribute>
    <xsl:if test="name($oldDataType) = 'structRef' or name($oldDataType) = 'enumRef'">
      <xsl:attribute name = "name" >
        <xsl:value-of select="$oldDataType/@name" />
      </xsl:attribute>
    </xsl:if>
  </xsl:element>
  <xsl:if test="count($new//range) > 0">
    <xsl:element name ="newRange">
      <xsl:attribute name = "min" >
        <xsl:value-of select="$new//range/min/." />
      </xsl:attribute>
```

```
      <xsl:attribute name = "max" >
        <xsl:value-of select="$new//range/max/." />
      </xsl:attribute>
    </xsl:element>
  </xsl:if>
  <xsl:if test="count($old//range) > 0">
    <xsl:element name ="oldRange">
      <xsl:attribute name = "min" >
        <xsl:value-of select="$old//range/min/." />
      </xsl:attribute>
      <xsl:attribute name = "max" >
        <xsl:value-of select="$old//range/max/." />
      </xsl:attribute>
    </xsl:element>
  </xsl:if>
  <xsl:if test="$oldDataTypeNodeStr != $newDataTypeNodeStr">
    <xsl:element name="dataTypeTreeChanged" />
  </xsl:if>
</xsl:template>


</xsl:stylesheet>
```

Appendix C: DTD/schema Formats

This appendix contains the DTD-definitions of the different MIB XML-file formats that were evaluated.

CCM IRP

This version of the CCM IRP DTD is defined in the specification document: Common Configuration Management N-Interface (N-IF), Specification: CORBA/XML Solution Set, ERA/RT-00:506, version 1, 2000-09-18, A.

```
<!--
Import/export file DTD.

-->

<!ELEMENT configDataCollection (fileHeader?,
                                configData,
                                fileFooter?)>


<!-- fileHeader -->

<!ELEMENT fileHeader EMPTY>
<!ATTLIST fileHeader fileFormatVersion CDATA #REQUIRED
                     senderName CDATA #REQUIRED
                     vendorName CDATA #REQUIRED>


<!-- configData -->

<!ELEMENT configData (managedObject*)>
<!ATTLIST configData mimName CDATA #REQUIRED
                     mimVersion CDATA #REQUIRED>


<!ELEMENT vendorSpecific EMPTY>

<!ELEMENT managedObject (modifier?, attribute*, vendorSpecific?)>
<!ATTLIST managedObject distinguishedName CDATA #REQUIRED>


<!-- modifier
The modifier is used only when downloading configuration
towards the managed sub-network.
-->
<!ELEMENT modifier (create | delete | update)>
<!ELEMENT create EMPTY>
<!ELEMENT delete EMPTY>
<!ELEMENT update EMPTY>



<!-- attribute -->

<!ELEMENT attribute ((structValue |
                     sequenceValue |
                     simpleValue |
                     referredMO |
                     undefinedValue), vendorSpecific?)>
```

```
<!ATTLIST attribute name CDATA #REQUIRED>

<!--
A simpleValue is a plain attribute value that is expressed
as a string. This is relevant for the following datatypes :
    * boolean can have the values "true" or "false"
    * long max and minimum values as defined in CORBA/IDL
    * float where Sign, FloatingPointLiteral are as
      defined in paragraph 3.10.2
      of the Java Language Specification. I
    * string as defined in CORBA/IDL
    * longlong max and minimum values as defined in
      CORBA/IDL
    * enum is represented as the long (as defined in
      CORBA/IDL) value of the attribute.
      The symbolical values of each enumerated values is
      defined in respective attribute definition in the NRM.

-->
<!ELEMENT simpleValue (#PCDATA)>

<!ELEMENT undefinedValue EMPTY>


<!-- structValue -->
<!ELEMENT structValue (structMember+ | undefinedValue)>
<!ATTLIST structValue name CDATA #REQUIRED>

<!ELEMENT structMember (simpleValue+ |
                        referredMO+ |
                        sequenceValue+ |
                        structValue+ |
                        undefinedValue)>
<!ATTLIST structMember name CDATA #REQUIRED>

<!-- sequenceValue -->
<!ELEMENT sequenceValue (simpleValue* |
                         referredMO* |
                         structValue* |
                         undefinedValue)>


<!-- referredMO -->
<!ELEMENT referredMO (null | distinguishedName)>
<!ELEMENT null EMPTY>

<!ELEMENT distinguishedName EMPTY>
<!ATTLIST distinguishedName distinguishedName CDATA #REQUIRED>


<!-- fileFooter -->

<!ELEMENT fileFooter (DateTime)>

<!ELEMENT DateTime (Date, Time)>

<!ELEMENT Time EMPTY>
<!ATTLIST Time
 Hour CDATA #REQUIRED
 Minute CDATA #REQUIRED
 Second CDATA #REQUIRED>
```

```
<!ELEMENT Date EMPTY>
<!ATTLIST Date
  Year CDATA #REQUIRED
  Month CDATA #REQUIRED
  Day CDATA #REQUIRED>
```

CCM IRP Example

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

<configDataCollection>
  <fileHeader vendorName="Ericsson" fileFormatVersion="1"
senderName="172.31.24.5"/>
  <configData mimVersion="1" mimName="CCM_NRM">
    <managedObject distinguishedName="Network=21000005">
      <attribute name="UserLabel">
        <simpleValue>Ericsson subnetwork</simpleValue>
      </attribute>
      <attribute name="NetworkType">
        <simpleValue>NEM domain</simpleValue>
      </attribute>
    </managedObject>
    <managedObject
distinguishedName="Network=21000005,ManagementNode=21000005">
      <attribute name="ManagementNodeType">
        <simpleValue>UTRAN-GSM-NEM</simpleValue>
      </attribute>
      <attribute name="UserLabel">
        <simpleValue>EricssonNEM</simpleValue>
      </attribute>
      <attribute name="LocationName">
        <simpleValue>Koeln_Office</simpleValue>
      </attribute>
      <attribute name="ManagementNodeVendor">
        <simpleValue>Ericsson</simpleValue>
      </attribute>
    </managedObject>
</configData>
  <fileFooter>
    <DateTime>
      <Date Day="1" Year="2000" Month="1"/>
      <Time Second="33" Minute="33" Hour="19"/>
    </DateTime>
  </fileFooter>
</configDataCollection>
```

VXML

VXML has two different DTD-definitions: Fundamental DTD (vxml) and Language DTD (vxmllang)

Vxml and vxmllang are effectively the same relative to the definition of XML elements and attributes. The key difference is in their usage. The vxmllang DTD makes extensive use of the composite element to show language structure. Aside from providing a more natural view (from the language's point of view) of the object structure, it permits attributes to be named by their language names rather than their database names. The vxml DTD "flattens" a database object, reflecting its database representation.

Fundamental DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT   vxml          (inst)* >
<!ELEMENT inst          (composite|attr)* >
  <!ATTLIST inst
            class         CDATA #REQUIRED
            id            CDATA #IMPLIED >
<!ELEMENT composite     (composite|attr)* >
  <!ATTLIST composite
            name          CDATA #REQUIRED >
<!ELEMENT attr          (#PCDATA) >
  <!ATTLIST attr
            name          CDATA #REQUIRED >
```

Language DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT   vxmllang      (inst)* >
  <!ELEMENT composite     (composite|attr)* >
<!ELEMENT inst          (composite|attr)* >
  <!ATTLIST inst
            class         CDATA #REQUIRED
            id            CDATA #IMPLIED
            hash          CDATA #IMPLIED >
<!ATTLIST composite
            name          CDATA #REQUIRED
            serialization CDATA #IMPLIED >
<!ELEMENT attr          (#PCDATA) >
  <!ATTLIST attr
            name          CDATA #REQUIRED
            serialization CDATA #IMPLIED >
```

Examples

The examples below are based on the following class-structures:

Java

```
class human {
    String    name;
    int       age;
}

class employee extends human {
    int       salary;
    int       daysOff[2];
}
```

Fundamental DTD

```
<?xml version="1.0"?>
<!DOCTYPE vxml SYSTEM
"http://www.versant.com/developer/vxml/dtds/v1.0/vxml.dtd">
<vxml>
  <inst class="employee" id="3.1.12345">
    <attr name="name">Fred</attr>
    <attr name="age">32</attr>
    <attr name="salary">50000</attr>
```

```
    <composite name="daysOff">
      <attr name="[0]">6</attr>
      <attr name="[1]">7</attr></composite></inst></vxml>
```

Language DTD

```
<?xml version="1.0"?>
<!DOCTYPE vxmllang SYSTEM
"http://www.versant.com/developer/vxml/dtds/v1.0/vxmllang.dtd">
<vxmllang>
  <inst class="employee" id="3.1.12345">
    <composite name="human">
      <attr name="name">Fred</attr>
      <attr name="age">32</attr></composite>
    <attr name="salary">50000</attr>
    <composite name="daysOff">
      <attr name="[0]">6</attr>
      <attr name="[1]">7</attr></composite></inst></vxmllang>
```

XMI

The XMI-specification is too complex to be included in this appendix. The latest specification can be found at http://www.omg.org/.

Mib.dtd

Due to demands from Ericsson the mib.dtd format has not been included in the public version of the thesis. The information has been moved to a company internal appendix, which only the examiner and Ericsson employees are allowed to access.

LINKÖPINGS UNIVERSITET

Avdelning, institution
Division, department

Institutionen för datavetenskap

Department of Computer
and Information Science

Datum
Date

2001-06-08

Språk
Language

[ ] Svenska/Swedish
[X] Engelska/English
[ ] _____

Rapporttyp
Report category

[ ] Licentiatavhandling
[X] Examensarbete
[ ] C-uppsats
[ ] D-uppsats
[ ] Övrig rapport

_____

ISBN —

ISRN —

Serietitel och serienummer **ISSN** —
Title of series, numbering

LiTH-IDA-Ex- 01/61

URL för elektronisk version

Titel
Title

Using XML for Import and Export of Data

Författare
Author

Martin Axlid

Sammanfattning
Abstract

Ericsson is developing a Corba service for data storage of radio networks. This service is implemented on top of an object database. The database contains data that describes a model of the physical network and its configuration. One task is to import and export the configuration data. Today XML is used as the file-format for the import and export. The current implementation of the import/export function has a linear growth of heap-memory consumption when the XML-files are processed. This causes the possibility of a fatal error when large amount of data should be handled. The purpose with the first part of the thesis has been to study and compare alternative XML-parsing techniques with limited memory consumption. The study shows that the best solution would be to use a combination of a SAX and DOM-parser in the import, and a non-standard “hardcoded” solution in the export.
Another task is to migrate data from one network model format to another; this is today performed outside the service. This can be very time-consuming, especially when the network model contains many elements, and there is therefore a need to make the process fully- or semi-automatic. The purpose of the thesis’s second part has been to find a suitable technique to perform the conversion. The study shows that an implementation of a new conversion tool in Java will be most effective and flexible. The use of a standard XML-conversion technique like XSL or a third party product would be less effective.
There is a need to make the format of the XML-file as effective as possible with respect to the following factors: correct functionality, easy implementation, simple readability and good runtime performance. In the third part of the thesis, the current format has been compared to several other “standard” XML-formats. The conclusion of this study is that the other formats do not have any significant advantage over the current format. The best solution would be to apply some minor changes to the current format and continue to use that.

Nyckelord
Keywords

XML, SAX, DOM, Parser, XSL, Java, Radio network, Object oriented database.