

Uppsala Master's Theses in
Computing Science 173
Examensarbete EIP
2000-09-06
ISSN 1100-1836

XML based interfaces to TelORB

Olle Johannesson
Sasan Tajallaei

Information Technology
Computing Science Department
Uppsala University
Box 311
S-751 05 Uppsala
Sweden

This work has been carried out at
Ericsson Utvecklings AB
Box 1505
125 25 Älvsjö
Sweden

Supervisor: Martin Sköld
Examiner: Sven-Olof Nyström

Passed:

Abstract

The requirements on future telecommunication platforms include high performance, fault tolerant, open architectures and scalability. TelORB, which is a distributed operating system for large-scale, embedded, real-time applications, fulfils these requirements.

This thesis consists of an evaluation of how XML (Extensible Mark-Up Language) can be used together with TelORB and implementation of a working prototype. The main task is to design an XML-based interface for analysing, populating and updating a TelORB database.

XML has been enthusiastically embraced in many application domains because there are a lot of applications that need to store data intended for human use as well as computerised manipulation.

An Application is presented that generates XML documents from a TelORB database backup file. The application uses an available program (Backupformatter) that translates the original backup file to the DBN external format, which is further translated in to XML code by means of a Java-parser.

TABLE OF CONTENTS

Chapter 1

Introduction.....	5
1.1 TelORB and XML.....	5
1.2 The Project.....	6

Chapter 2

TelORB.....	7
2.1 History.....	7
2.2 What is TelORB?.....	7
2.3 TelORB Characteristics.....	8
2.3.1 <i>Open System</i>	8
2.3.2 <i>Fault Tolerant</i>	9
2.3.3 <i>High Performance</i>	9
2.3.4 <i>Object Oriented</i>	9
2.4 TelORB System Overview.....	9
2.4.1 <i>Communication Layer</i>	10
2.4.2 <i>Database Layer</i>	10
2.4.3 <i>Software Management Layer</i>	10
2.4.4 <i>CORBA-ORB module</i>	10
2.5 The Database (DBN).....	11
2.6 Different ways of accessing database instances.....	11
2.6.1 <i>Keyed access</i>	11
2.6.2 <i>Access by reference (DID)</i>	11
2.6.3 <i>Access by class extension</i>	11
2.7 TelORB execution model.....	12
2.8 TelORB today.....	13

Chapter 3

Introducing XML.....	14
3.1 Abstract.....	14
3.2 XML, SGML and HTML.....	14
3.2.1 <i>SGML</i>	14

3.2.2	<i>HTML</i>	15
3.2.3	<i>XML</i>	15
3.3	Meta-Markup Language.....	15
3.4	XML separates definition from content.....	16
3.5	Editors, Parsers and Browsers.....	16
3.6	Validity.....	16
3.6.1	<i>DTD</i>	16
3.6.2	<i>XML Schemas</i>	16
3.7	Style Sheets.....	16
3.7.1	<i>Cascading Style Sheet</i>	17
3.7.2	<i>Extensible Style Language</i>	17
3.8	URLs and URIs.....	17
3.9	Links and Pointers.....	17
3.9.1	<i>XLinks</i>	17
3.9.2	<i>XPointers</i>	18
3.10	XQL (XML Query Language).....	18
3.11	XML and Java.....	18
3.12	Summery.....	18

Chapter 4

The Assignment.....19

4.1	Understanding the assignment and getting familiar with TelORB and XML.....	19
4.2	The main task.....	20
4.2.1	<i>Parser</i>	21
4.2.2	<i>Program Execution</i>	21
4.2.3	<i>Producing XML document</i>	23
4.2.4	<i>Composing the DTD-document</i>	23
4.2.5	<i>The CSS (Cascading Style sheet)</i>	24
4.2.6	<i>XML-document</i>	24
4.3	Displaying the files.....	25
4.4	Running the application.....	26
4.4.1	<i>Test-run (Searching for a class)</i>	26
4.3.1	<i>Test-run (Searching for a class with a specific key)</i>	28

Chapter 5

Conclusions and future work.....30

Assorted Information

Dictionary.....	32
References.....	34
TBSE Users Guide.....	Appendix A
DBN External Format Guide.....	Appendix B

INTRODUCTION

The requirements on future telecom and datacom platforms include high performance, fault tolerance, open architectures, and scalability.

1.1 TelORB and XML

TelORB [1] is a distributed operating system for real-time applications that require a robust platform with 100% availability. It consists of a modern OS kernel, an Object-Oriented real-time database management system, and flexible mechanisms for controlling the software configuration with support for online software upgrade without compromising the system availability. To support flexible application development and open interfaces TelORB also includes a Java Virtual Machine (JVM) [2] and a CORBA-compliant Object Request Broker [3].

XML stands for eXtensible Mark-up Language [4]. It is a simplified subset of a previous mark-up language standard called SGML (Standard Generalised Mark-up Language) [5], and was devised by a committee of the World Wide Web consortium [6] in response to the need for a generalisation of HTML, the Hypertext Mark-up Language [7] used for formatting Web pages. SGML was designed by ISO (the International Standards Organisation) [8] as a new standardised mark-up language that enshrined this separation of structure and presentation. In order to apply SGML one creates a Document Type Definition (DTD) [4] that defines the set of valid tags for the documents being created, and uses DSSSL (the ISO-standardised Document Style Semantics and Specification Language that accompanies SGML) [8] to define how to display text labelled with those tags.

XML has being enthusiastically embraced in many application domains because there are a lot of applications that need to store data intended for human use as well as computerised manipulation. One example might be storing and displaying telecom subscriber information. An XML DTD for storing subscriber data makes it comparatively easy to write applications that can read subscriber data without mixing up different fields, such as subscriber numbers, subscriber services, and billing data. For each application you simply need to define a DTD for the tags and how they can be legally combined. These DTDs can then be used with XML parsers and other XML tools to rapidly create applications to process and display the stored information in whatever way is required.

XML is intended for the storage and manipulation of text making up human-readable documents like Web pages, while middleware solutions like CORBA tie together cooperating computer applications exchanging transient data that will probably never be directly read by anyone. Neither of these technologies will replace the other, but instead they will increasingly be used together.

The current external on-line interfaces to TelORB are all based on CORBA and graphical tools for controlling the configuration exist. All interfaces are here described as CORBA IDL [1] specifications.

The database schema in TelORB is specified through meta-data that describes the classes in the current database, their inheritance structure, and their attributes. In the database this meta-data is available through a special meta-class that describes the existing classes and the corresponding attributes. The meta-data is also available through an export format that makes it possible to analyse backup files off-line. An import tool is being developed that should be able to import data defined by the export data to make off-line populating possible.

1.2 The Project

This thesis consists of an evaluation of how XML can be used together with TelORB and also implementation of a working prototype.

The main work is to design an XML-based interface for analysing and populating a TelORB database. To support browsing and generating databases through XML a DTD has to be defined for the TelORB database meta-data. This DTD can then be used for browsing a populated database and to support generating XML-files that are translated into the export format that can be imported into TelORB.

The prototype is implemented in Java [9] where XML, DTD and CSS (Cascading Style Sheet) [4] files are generated from the database content by reading data in the export format and displaying it in an XML-browser.

The first step in performing this thesis was to become familiar with TelORB and XML. The second step was to find an XML editor and an XML browser that could be used during the project. When these two steps were accomplished successfully the main work could be executed.

Chapter 2

TelORB

This chapter is an a short presentation of the TelORB operating system that aims to give some knowledge about TelORBs history, characteristics and system overview. Most of this information is gathered from the TelORB Software Implementation handbook with only some minor changes to make sure that the reader gets an accurate description of the system.

2.1 History

TelORB arose from the remains of a major project that was intended to develop a completely new distributed platform. The operating system performance was found to be better than anticipated, and a group of persons were therefore allowed to continue development on a small scale when the larger project was discontinued. That was in 1996. Four years later, millions of subscribers are using systems based on the new platform.

Today's operators must address a number of problems. Network nodes must never go down, and performance demands are also increasing constantly. Operators therefore want to be able to build new systems without knowing from the start how large they will become or how much capacity will be needed. They want to be able to continue network expansion while incorporating the latest technology without having to replace existing systems, and they want to be able to buy nodes from different suppliers. The new operating system meets their demands.

2.2 What is TelORB?

TelORB is a distributed operating system suitable for controlling telecom applications. It can be loaded onto a group of processors (see figure 2.1). The group of processors will behave like one single system. Applications can run on top of TelORB. Different parts of the application can run on different processors. One part can communicate with another in a transparent manner. If there is a need to increase the processing capacity you can add more processors in run-time without disturbing ongoing activities. The increase of the capacity is a linear function of the number of processors. Therefore TelORB is described as a truly scalable system.

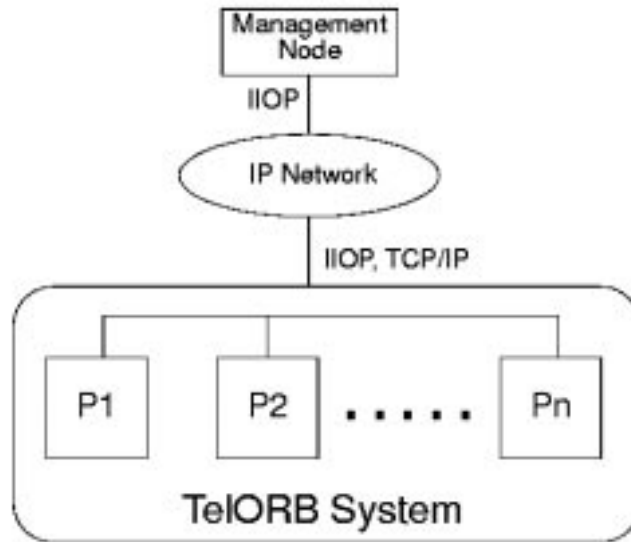


Figure 2.1 The group of processors forming the TelORB system can communicate with the outside world through IIOp and TCP/IP.

2.3 TelORB Characteristics

TelORB works with several different commercial technologies. Advances in data technology are rapid, and when new products become available, operators should be able to purchase new hardware without replacing entire systems. The demand for open interfaces is also satisfied in that the system supports several standard-programming languages. Newly recruited developers can thus become immediately productive without first having to go through time-consuming training in a new language. Linear scalability allows operators to start on a small scale and then grow at their own pace. Put simply, if there are two processors and capacity needs to be doubled, the operator just adds two more processors. If four are not enough, more can be added as required, rather than having to replace old processors with new ones. So apart from being distributed and scalable TelORB can also be described as: Open, Fault Tolerant, High Performance and Object Oriented.

2.3.1 Open System

TelORB is an open system in several ways.

- The external protocols of the system are TCP/IP [10] and IIOp (Internet Inter-Orb Protocol) [10], see figure 2.1 above. TelORB includes a so-called ORB implementing the IIOp, hence the name TelORB. IIOp is a part of a standard called CORBA (Common Object Request Broker Architecture). IIOp is also used for managing the TelORB system.
- The processors used (Pentium or Sparc) are commercially available.
- Applications can be programmed in C++ [11] or Java.

2.3.2 Fault Tolerant

It is also important that these kinds of systems always keep running. With TelORB, if one processor fails, its tasks are automatically distributed among the remaining processors. In the same manner, other processors take over if maintenance is needed in one part of the system or if hardware is replaced. Delivering real-time performance under these conditions, however, can be difficult. Systems with extremely high availability tend to be slow, but with TelORB excellent results have been achieved.

A properly designed TelORB application is extremely fault tolerant. The fault tolerance is achieved primarily by duplication. Functionality and persistent data is duplicated on at least two processors. The external interfaces of the system (Ethernet) are duplicated as well. There is no single point of failure in a TelORB system. It means that one fault alone (e.g. a processor crash) will not reduce the service availability of the system.

2.3.3 High Performance

The linear scalability makes TelORB capable of handling large nodes requiring high performance (Fig 2.2).

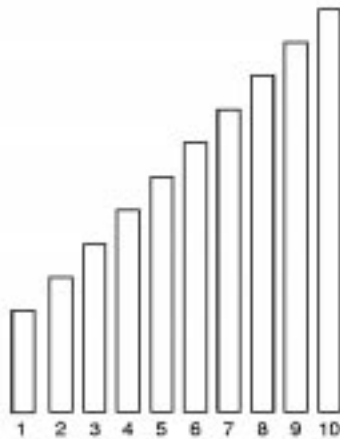


Figure 2.2 Capacity grows as processors are added

2.3.4 Object Oriented

The programming languages C++ and Java are object oriented. Objects are specified in Delos [1] and IDL specification languages. Interface Definition Language (IDL) is part of the CORBA standard.

2.4 TelORB System Overview

Figure 2.3 gives an overview of the hardware in TelORB. As shown there are a number of processors, 4 in this example, interconnected through Ethernet switches (not represented in the picture).

These processors are connected to an I/O processor, a UNIX machine, used to perform I/O operations like initial loading and dumping.

The Kernel of the TelORB operating system is distributed in the central memory of each processor.

The TelORB operating system provides a structure divided in four layers:

1. Communication Layer
2. Database Layer
3. Software Management Layer
4. CORBA-ORB module

2.4.1 Communication Layer

The Communication Layer provides IPC, Inter Process Communication mechanism.

2.4.2 Database Layer

The Database Layer provides a robust database for storage of POT (Persistent Object Type) [1]. The database is always held in primary memory, and distributed over all the processors. Fault tolerance is achieved storing each piece of data on more than one processor, so even with a failing processor database is still available to the application.

2.4.3 Software Management Layer

The Software Management Layer automatically configures the executing software so that it runs efficiently on the processors that are available to the TelORB system.

2.4.4 CORBA-ORB module

The CORBA-ORB module provides interoperability between applications on different machines in heterogeneously distributed environments and seamlessly interconnects multiple object systems.

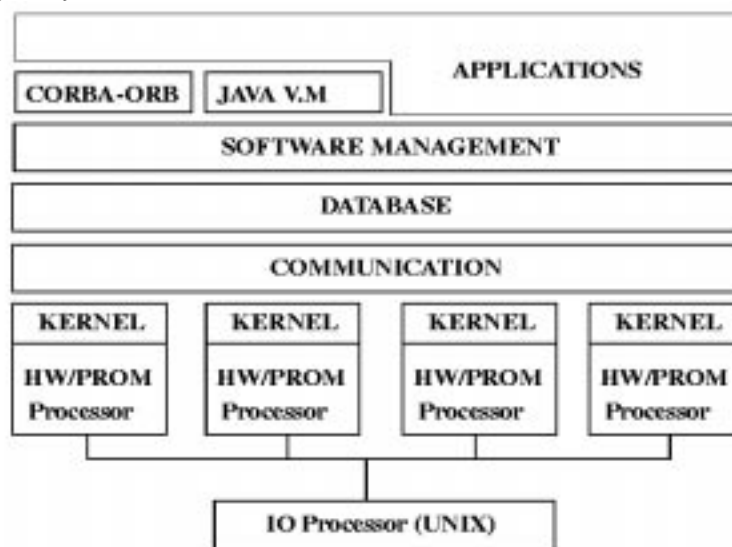


Figure 2.3 TelORB system overview

2.5 The Database (DBN)

While data inside a process is volatile (it is lost if the process or the processor it runs on crashes), data stored in the TelORB database persists even after a process or processor crashes. Besides storing data, the database shares data between processes. The TelORB database is an object-oriented, real-time database that stores data in primary memory on the processors.

The database is used to store:

- long-lived data (longer than the processes using it)
- data that must be stored safely (e.g. charging data)
- data that must be consistent
- data to be shared by several processes

2.6 Different ways of accessing database instances

There are three ways to access database instances (see figure 2.4 below).

1. Keyed access
2. Access by reference
3. Access by class extension

2.6.1 Keyed access

When specifying a database object in Delos you can choose to give the object a primary key. For example a Subscriber object could have subscriber number as primary key. An instance can be created/accessed by calling methods in the Subscriber object, passing the key as an argument. The database converts the key into a global database reference called the DID (Database Identifier) [1]. It is unique for this database instance within the TelORB system.

2.6.2 Access by reference (DID)

Not all database objects are specified to have a primary key. For example the Sub_Profile object of figure 2.4 below has no primary key. But the keyed Subscriber object includes a reference to a Sub_Profile object. This attribute is the DID of the Sub_Profile instance. By first reading this attribute, the Sub_Profile instance can be accessed by calling the Sub_Profile object, with the DID as argument. Access by reference should always be used if possible since it is faster than access by key.

2.6.3 Access by class extension

It is possible to iterate over all instances of the same type e.g. Subscribers. Conditions can be specified for the iteration. Only the objects fulfilling the conditions are opened.

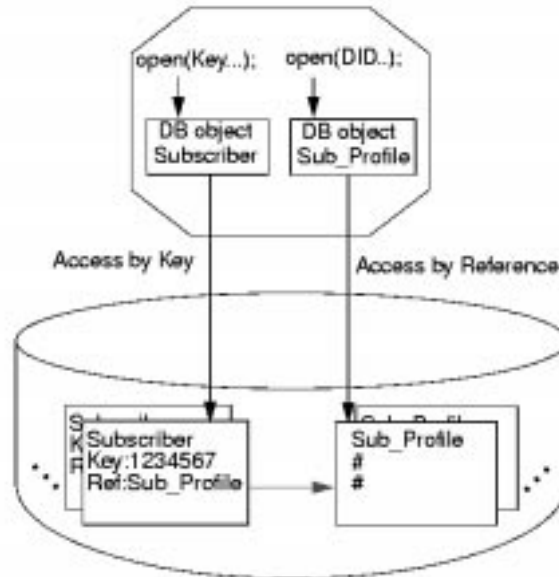


Figure 2.4 Database instances can be accessed by key or by reference (DID).

2.7 TelORB execution model

In TelORB a thread is an event receiver, the fundamental execution principle is that a process has one thread that reacts to events (such as dialogue invocations, timers, etc) by executing call-back functions in objects associated with the events, see figure 2.5.

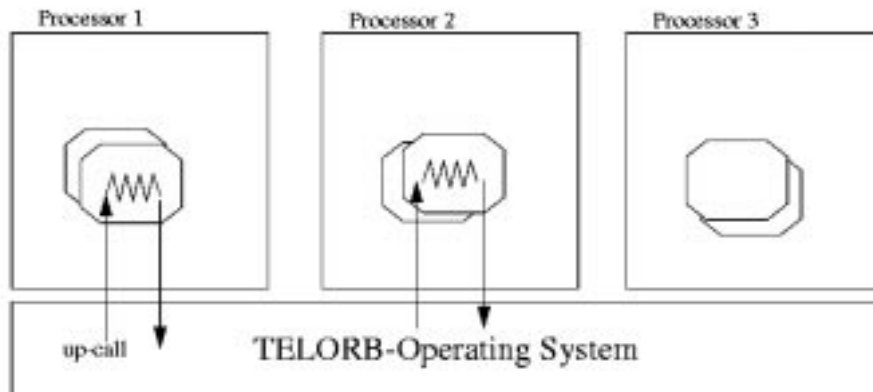


Figure 2.5 Each process has one thread that reacts to events

The handling of these events get serialized so the thread never reacts to an event until the execution associated with the previous event is finished, i.e. the call-back function returns. The reason is that TelORB system is asynchronous by its nature, events are implemented with so-called up-calls (an up-call is when the system calls an user-defined method), the called method must always return to the operating system.

In this way the thread is never blocked in a call, it initiates a task, receiving an up-call, and returns with a callback, with the clear advantage that events get handled in "real time".

As the TelORB thread must return from its callback function in order to handle the next event, the normal Java paradigm, where threads wait for events in the middle of their execution paths (inside monitors), cannot readily be used.

TelORB therefore defines two kinds of threads in Java, the "process thread", that is able to handle TelORB events, and the normal Java thread. Processes can have several process threads, but an event is always handled by the thread that was executing when the object associated with the event was created; this can lead to some surprises. The first, and possibly only, thread starting in a process is of course a process thread.

2.8 TelORB today

The market for intelligent network solutions is growing rapidly, which means that there will be an increased need for zero-downtime platforms that support a massive amount of transaction-oriented processing. Nodes in intelligent networks often need ultra-fast databases to handle requests from a large number of users. The TelORB platform is well suited to meet these requirements. Furthermore, its exceptional scalability permits operators to gradually expand their systems as the need arises.

Because the system uses commercially available, "off-the-shell" processor boards, operators can always take advantage of the latest achievements in hardware design.

Applications are built using well-known languages, such as C++ and Java, and interoperability is provided through the built-in object request broker.

TelORB is an open platform whose characteristics, in terms of robustness and flexible configurations, are unparalleled. It is currently deployed as the base for the Jambala platform, which was placed in commercial operation in Southwestern Bell's TDMA network in Chicago in 1999. Today there are more than 8 million subscribers widely spread over North and South America using TelORB and it is estimated that the number will increase to 40 million by the end of year 2000.

Introducing XML

In this chapter we take a look at the history and theory behind XML [4] and the goals XML is trying to achieve. We will also look closer at how the different pieces of the XML equation fit together to create and deliver documents to readers.

3.1 Abstract

XML stands for Extensible Markup Language and is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. XML is very flexible and can be used for many different purposes.

3.2 XML, SGML and HTML

XML is a subset of SGML (Standard Generalised Markup Language) [5] specially designed for Web applications. This subset retains the key SGML advantages of extensibility, structure and validation in a language that is designed to be as simple as HTML (Hypertext Markup Language) [6] to use.

3.2.1 SGML

SGML is a well-established meta-markup language that was developed to express structure and content in different types of complex electronic documents. For a couple of reasons SGML has never reached widespread public usage. One is that it has been considered to be too complex. Another one is that it is not designed to be used over the Internet. Here are some SGML characteristics:

SGML – Advantages

- Document is separated from presentation
- User-defined tags
- Well-established

SGML – Disadvantages

- Must have DTD to be able to see the data
- Complicated to learn
- Not supported by popular browsers

3.2.2 HTML

To get around the complexity of SGML Tim Berners-Lee developed HTML in 1990. Today HTML is very popular and is used for creating Web pages that can be viewed all around the world. HTML was designed to be as simple as possible to use, which however resulted in some restrictions. HTML characteristics can be summarized according:

HTML – Advantages

- Simple – fixed set of tags
- Portable – can be used with different software

HTML – Disadvantages

- Limited tag set
- Difficult to read and understand the code
- Can not separate the definition from content
- Can not define a structure of contents

3.2.3 XML

In an attempt to combine the strength of SGML and HTML without their restrictions W3C (World Wide Web Consortium) developed XML with the characteristics given below. It should be noted that XML and associated specifications are still under development and new drafts are issued almost every month.

XML – Advantages

- Platform and system independent
- User-defined tags
- Does not require explicit DTD
- Display format and content are separated

XML – Disadvantages

- “Pickier” than HTML
- Must be converted to HTML to be viewed in all browsers today

3.3 Meta-Markup Language

XML is a meta-markup language, which means that it gives you can define your own tags that are specific to a particular purpose, and have the possibility to describe information consistently. These tags must be organized according to certain general principles, but they are quite flexible in their meaning. This gives you the advantage of not having to wait for browser vendors to catch up with what you want to do. You can invent the tags you need, when you need them, and tell the browsers how to display these tags.

3.4 XML separates definition from content

XML markup describes a documents structure and meaning. It does not describe the formatting of the elements on the page. Formatting can be added to a document with a style sheet. The document itself only contains tags that say what is in the document, not what the document looks like.

3.5 Editors, Parsers and browsers

XML documents are most commonly created with an editor. This may be a basic text editor like Notepad or it may be a structured editor that displays XML documents as trees. These editors normally have build-in parsers that read the document and verify that the XML it contains is well formed. It may also check that the document is valid, though this test is not required. If the document passes the test, the processor converts the document into a tree of elements.

Finally the parser passes the tree or individual nodes of the tree to the end application. This application may be a browser or some other program that understands what to do with the data. If it is a browser, the data will be displayed to the user. But other programs may also receive the data. For instance, the data might be interpreted as input to a database.

3.6 Validity

Individual documents can be compared against DTD or schema in a process known as validation. If the document matches the constraints listed in the DTD or schema, the document is said to be valid. If it does not, the document is said to be invalid.

3.6.1 DTD

DTD stands for Document Type Definition and is used to define a grammar and to specify a set of rules for the structure of a document. It provides a list of the elements, attributes, notations and entities contained in a document, as well as their relationships to one another. DTDs can be included in the file that contains the document they describe, or they can be linked from an external URL.

3.6.2 XML Schemas

XML Schemas is a much richer and more extensible way to describe the rule for the content of a document. There is for example a greater support for different data-types in XML Schemas than in DTD. Furthermore you are able to define your own data-types as you need them.

3.7 Style Sheets

Since XML allows arbitrary tags to be included in a document, there is not any way for the browser to know in advance how each element should be displayed. When you

send a document to a user you also need to send along a style sheet that tells the browser how to format individual elements. This allows you to control the styling aspects of a document without compromising its structure.

3.7.1 Cascading Style Sheet

One kind of style sheet you can use is a Cascading Style Sheet (CSS). CSS, initially designed for HTML, defines formatting properties like font size, font family, font weight, paragraph indentation, paragraph alignment, and other styles that can be applied to particular elements. Multiple style sheets can be applied to a single document, and multiple styles can be applied to a single element.

3.7.2 Extensible Style Language

The XSL [4] is a more advanced style-sheet language specially designed for use with XML documents. XSL documents are themselves well-formed XML documents.

XSL documents contain a series of rules that apply to particular patterns of XML elements. An XSL processor reads an XML document and compares what it sees to the patterns in a style sheet. When a pattern from the XSL style sheet is recognized in the XML document, the rule makes the desired combination of outputs.

3.8 URLs and URIs

XML documents can live on the Web, just like HTML and other documents. When they do, they are referred to by Uniform Resource Locators (URL) [10], just like HTML files. Although URLs are well understood and well supported, the XML specification uses the more general Uniform Resource Identifier (URI) [10]. URIs are a more general architecture for locating resources on the Internet, that focus a little more on the resource and a little less on the location.

3.9 Links and Pointers

Since XML documents can be posted on the net, the ability to address them and use links between them are highly desirable. Standard HTML link tags can be used in XML and HTML documents can link to XML documents. However, XML lets you go further with XLinks [4] for linking to documents and Xpointers [4] for addressing individual parts of a document.

3.9.1 XLinks

XLinks enable any element to become a link. Furthermore, links can be bi-directional, multidirectional, or even point to multiple mirror sites from which the nearest is selected.

3.9.2 XPointers

XPointers enable links to point not just to a particular document at a particular location, but to particular part or element of a particular document. This way XPointers provide extremely powerful connections between individual pieces of documents.

3.10 XQL (XML Query Language)

XQL [4] is a way to locate and filter the elements (data fields) and text in an XML document. XQL provides a tool for finding and/or selecting out specific items in the data collection in an XML file or set of files. It is based on the pattern syntax used in the XSL and is proposed as an extension to it

3.11 XML and Java

XML and Java technologies have many complementary features and when used in combination they enable a powerful platform for sharing and processing of data and documents.

3.12 Summary

XML is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications such as databases, e-commerce, Java, web development, searching, etc. The main benefits with XML can be divided into three major points:

- **Searching and Information Exchange**
With common XML vocabularies searches can be more meaningful and applications can share data more intelligently.
- **Web Application Development**
Web applications will be able to use XML for local processing, to prevent multiple views of data and represent more complex data structure.
- **Distributed Computing**
XML can effectively future-proof data formats, exchange data structures and enhance Web documents into robust platforms for system integration.

XML and TelORB

As mentioned in chapter 1.2 the propose with this thesis is to evaluate how XML can be used together with TelORB by designing an XML-based interface for analysing and populating a TelORB database. In order to understand the assignment and to be able to solve it we divided it into several manageable parts. The first step is to get familiar with TelORB and XML and understand how they work. Then we have to find out how to combine these two in a sensible way.

4.1 Understanding the assignment and getting familiar with TelORB and XML

We started with attending a self-study course [1] to learn about TelORB. First we learned what TelORB is by reading about TelORBs characteristics, system overview, software basics and distribution of the database instances. The next step was to design our own static TelORB process. To learn how to compile and link the program code we wrote an executable loadmodule and loaded and started it on a Vega (simulated processor environment). The objectives with this exercise were to get familiar with TelORB design environment, to learn how to design and customize a static TelORB process and finally to load and run an application in the Vega environment using a gdb (GNU debugger, is used for high-level debugging of C and C++ code, at the present a debugger for Java is developed for TelORB).

The next exercise intended to give a basic understanding how to design an application using TelORB database. The main objective with this exercise was to learn how to design a database object and to implement read and write operations to this object. Since the results of this exercise are later used to illustrate how the program works, we decided to design a simple database that contains only a few numbers of objects. A real TelORB database consists of thousandths or even millions of objects.

The introduction to XML had to be done with out any help from a course, so the way to get information about XML was to read books, other documentation's regarding XML and to search on the Internet. One of the thesis objectives was to find an XML-browser that will be used to show the generated XML-document. An XML-editor was also needed to see if an XML-document is valid.

There are a lot of XML-browsers available, but there are only a few that actually work. That is because XML is still expanding. The browsers that work are DocZilla [12] form Netscape [13], MoZilla [14] from Netscape and Internet Explorer from Microsoft [15]. We decided to use MoZilla but after a while we discovered that this browser did not consider the DTDs. So Internet Explorer where used instead. That limited the use of this application to Windows NT/2000 [15] where MS Explorer will

open the resulting XML-files automatically, i.e. on Linux [16] and Solaris [17] these files have to be opened manually.

When it comes to an XML-editor, we found that XMLSpy [18] is an editor that functions for our purposes. XMLSpy controls if the XML-document is valid. With help of XMLSpy a hand coded XML-document can be checked if valid. That can be used to learn how an XML-document shall be build.

4.2 The main task

To make a backup of the database object that has been generated form the previous exercise, TelORB-browser is the tool to use. TelORB-browser is an application where you can see the running processors, add a processor, do software backup and make and use backups. All of this is done in the simulated processor environment named Vega, which is a UNIX [19] process with a hardware adaptation layer that acts as an ordinary processor.

The raw backup files are very tricky, almost impossible to understand (Fig. 4.1).

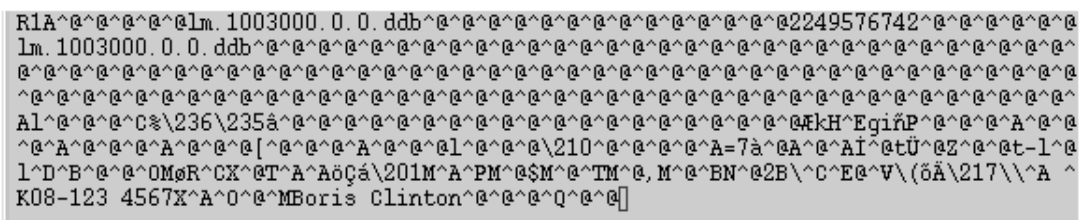


Figure 4.1 lm.1003000.0.0.ddb

A program called BackupFormatter can be used to make this more understandable. The BackupFormatter translates the original backup-file to the DBN External Format (see Appendix B for a detailed description). The DBN External Formats format is more structured and easier to understand (Fig. 4.2), but still the structure is very flat.

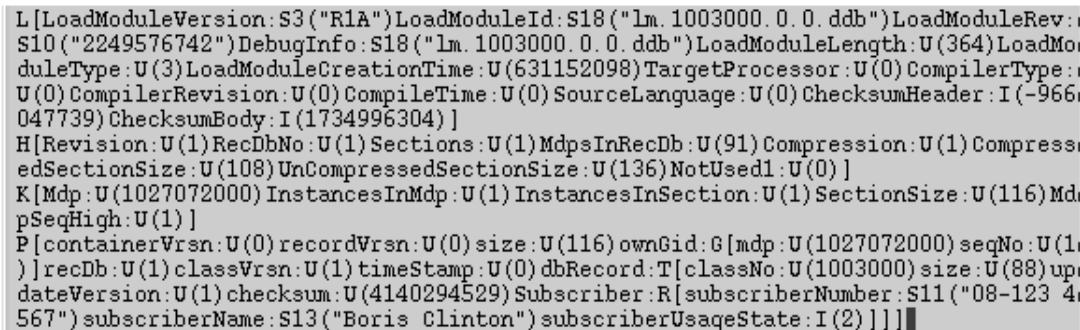


Figure 4.2 lm.1003000.0.0.ddbe

4.2.1 Parser

From this it was obvious that our problem could be solved in two ways. First, to modify the source-code in the BackupFormatter, which is written in C++, and generate XML-documents. Second, to make our own parser that reads through the ddbbe-files and then calls different functions, which generates the XML-code (Fig. 4.3). Since these functions should be written in Java we decided that the second way was a more attractive choice.

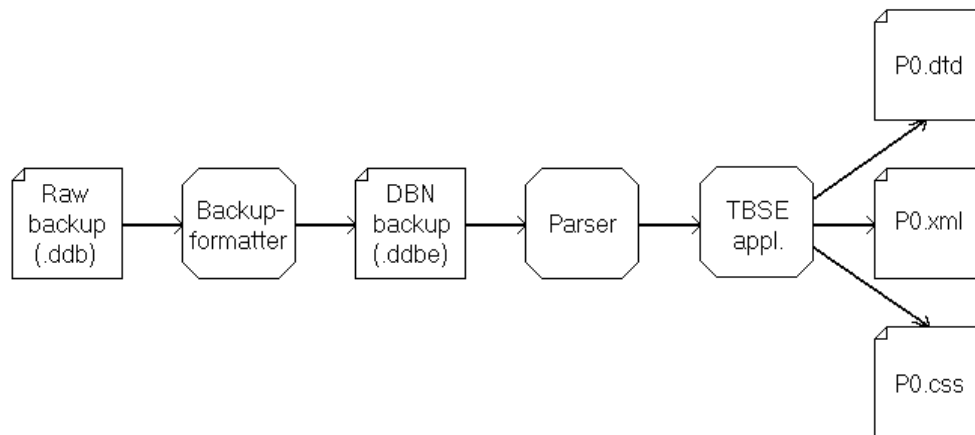


Figure 4.3 Export schematics

4.2.2 Program Execution

We decided that the user should be able to choose between either a search for a class or a class with a specific key from either a graphical GUI or the command shell.

The parser begins with reading through the DBN-backupformatted file "Im.1.0.0.0.ddbe" where all classes are represented and the name of the files in which the information for every class is stored can be found.

If the classname is not found then no code at all will be generated and an error-message will indicate that.

When a sought class is found, the filename is saved in an variable that later will be used when the parser gets its second call. The parser reads through this new file that contains information regarding the sought class. While the parser reaches different points in this file it will make a call to a Java function that will perform a specific task depending on where in the file the parser is reading from. E.g. if the parser reads a "P" then a call to the "setP"-function will be carried out which generates the XML-, DTD-, CSS- and HTML code (Fig 4.4).

If a specific key is sought, but the key does not exist in the class, only the header information regarding the class will be generated.

Regardless if any class was found or not the application returns to its starting point and gets ready for a new search.

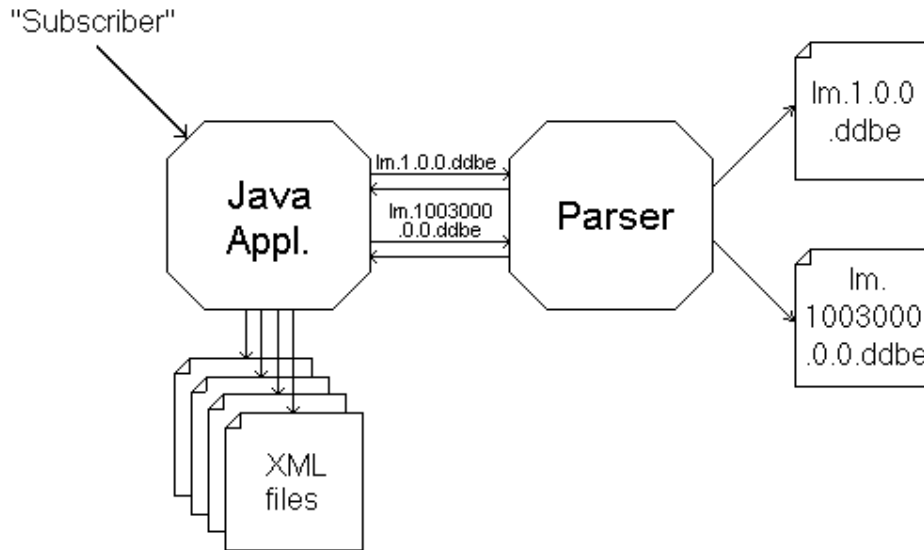


Figure 4.4 Application schematics

To make a parser you first have to define the grammar that the parser will follow when it reads through the input text. The grammar is the rules for the parser to follow. In the grammar you define the characters and numbers that are allowed to occur and in which order they should occur. An error message is therefore generated when the parser detects that the input text does not follow the defined grammar (Fig. 4.5).

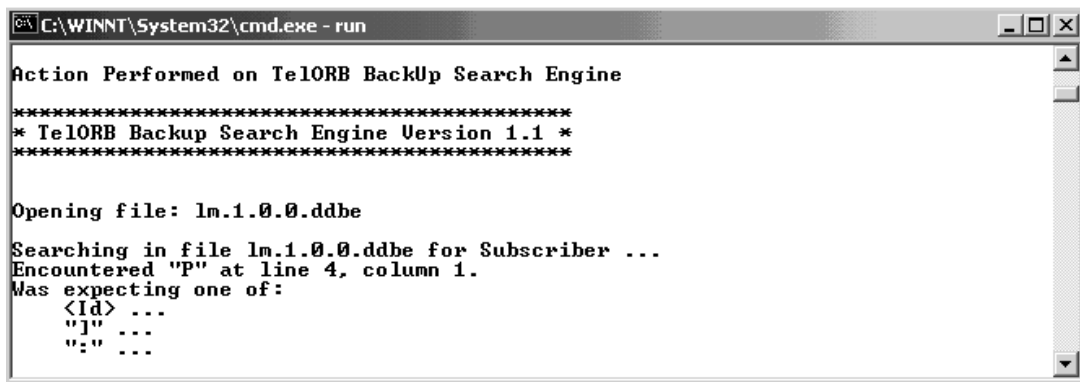


Figure 4.5 Parse Error

When the grammar is specified, a parser generator is needed to make the grammar run-able. A parser generator is a tool that reads the grammar specification and converts it to a program that can recognize matches to the grammar. In our case we wanted a parser that could run together with Java applications. JavaCC (Java Compiler Compiler) [20] is a parser generator for Java that originally was released by Sun Microsystems [17], Inc. Metamata has now taken over the distribution and support. JavaCC has been certified to be 100% Pure Java and runs on over 40 different platforms without any special porting requirements. That makes JavaCC portable.

4.2.3 Producing XML document

The first step is to model the document, to decide if a DTD or an XML-Schema should be used. The main difference between a DTD and an XML-Schema is that with the XML-Schema the data types are considered. We chose to use a DTD because it is easier to generate and when exporting the database object it is not necessary to consider the data types. XML-Schemas could be used later when an object shall be imported to the database.

4.2.4 Composing the DTD-document

When designing a DTD it is essential, that each tag is given a describing name so that the document itself becomes self-describing. In our case this is not a problem because each tag-name is taken from the database. So the DTD will be self-describing.

The DTD is build up of two main parts. The first part describes the one and only order that the tags can appear. Then follows the declaration part. Each tag-name is declared once. If a tag is declared more than once an error-message will be generated. To generate the DTD some function calls has to be done. The first part is generated while the parser reads through the backup-file and the name that the tags are given are saved in an array If a name already has appeared it will not be saved. Then it is time for the declaration part. The names that had been saved earlier are now used when the tags are being declared.

Since the DTD for the header part is always the same we decided to use a hand-coded DTD for this part and let the application generate new ones for every "Record" i.e. for the part that contains the specific information about the sought class. Below (Fig. 4.6) is an example of a generated DTD.

```
<!ELEMENT Record (Vaule3, containerVrsn, recordVrsn, size, mdp,
                  seqNo, recDb, classVrsn, timeStamp, classNo,
                  size, updateVersion, checksum, subscriberNumber,
                  subscriberName, subscriberUsageState, subscriberAge,
                  subscriberInitial, subscriberSex, Weight)>
<!ELEMENT containerVrsn (#PCDATA)>
<!ELEMENT recordVrsn (#PCDATA)>
<!ELEMENT mdp (#PCDATA)>
<!ELEMENT seqNo (#PCDATA)>
<!ELEMENT recDb (#PCDATA)>
<!ELEMENT classVrsn (#PCDATA)>
<!ELEMENT timeStamp (#PCDATA)>
<!ELEMENT classNo (#PCDATA)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT updateVersion (#PCDATA)>
<!ELEMENT checksum (#PCDATA)>
<!ELEMENT subscriberNumber (#PCDATA)>
<!ELEMENT subscriberName (#PCDATA)>
<!ELEMENT subscriberUsageState (#PCDATA)>
<!ELEMENT subscriberAge (#PCDATA)>
<!ELEMENT subscriberInitial (#PCDATA)>
<!ELEMENT subscriberSex (#PCDATA)>
<!ELEMENT Weight (#PCDATA)>
```

Figure 4.6 PO.dtd

4.2.5 The CSS (Cascading Style sheet)

We have used a style sheet for the presentation of the XML-document. In the style sheet you describe how each tag shall be presented such as font size, type, colour, etc.

Just like the DTD, the CSS for the header part is always the same so we use a hand-coded CSS for this part and let the application generate the CSS for the “Record” part. Below (Fig. 4.7) is an example how the CSS for the XML document that uses the DTD above (Fig. 4.6) looks like.

```
Record { display: block; text-indent: 2in; }
containerVrsn { display: block; }
recordVrsn { display: block; }
size { display: block; }
mdp { display: block; }
seqNo { display: block; }
recDb { display: block; }
classVrsn { display: block; }
timeStamp { display: block; }
classNo { display: block; }
size { display: block; }
updateVersion { display: block; }
checksum { display: block; }
subscriberNumber { display: block; }
subscriberName { display: block; }
subscriberUsageState { display: block; }
subscriberAge { display: block; }
subscriberInitial { display: block; }
subscriberSex { display: block; }
Weight { display: block; }
```

Figure 4.7 P0.css

4.2.6 XML-document

When generating an XML-document there are a few things to think about.

In the document header of each XML-document a path to the belonging DTD and CSS has to be declared so that the XML-document knows which DTD and style sheet that belongs to the document. The XML version and the text encoding type are also declared.

When using an XML-Schema or XSL a declaration in the header is not necessary, because the XML-Schema and the XSL is a development of the XML-language, unlike the DTD and CSS that both are remains of SGML.

The tag names that are being used must have been declared in the corresponding DTD so that the tags in the XML-document is generated in the right order according to the tag-order described in the DTD (Fig. 4.8).

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE DBNexternalFORMAT SYSTEM "PO.dtd">
<?xml-stylesheet href="stillmall.css" type="text/css"?>
<?xml-stylesheet href="PO.css" type="text/css"?>
<P>
<Value1>RecContainer</Value1>
  <containerVrsn>containerVrsn: 0</containerVrsn>
  <recordVrsn>recordVrsn: 0</recordVrsn>
  <size>size: 116</size>
<G>
<Value2>ownGid</Value2>
  <mdp>mdp: 1027072000</mdp>
  <seqNo>seqNo: 1</seqNo>
</G>
  <recDb>recDb: 1</recDb>
  <classVrsn>classVrsn: 1</classVrsn>
  <timeStamp>timeStamp: 0</timeStamp>
<T>
<Value2>DbRecord</Value2>
  <classNo>classNo: 1003000</classNo>
  <size>size: 88</size>
  <updateVersion>updateVersion: 1</updateVersion>
  <checksum>checksum: 4140294529</checksum>
<R>
<Value2>Record</Value2>
  <subscriberNumber>subscriberNumber: 06-123 4567</subscriberNumber>
  <subscriberName>subscriberName: Boris Clinton</subscriberName>
  <subscriberUsageState>subscriberUsageState: 0</subscriberUsageState>
  <subscriberAge>subscriberAge: 50</subscriberAge>
  <subscriberInitial>subscriberInitial: 66</subscriberInitial>
  <subscriberSex>subscriberSex: 1</subscriberSex>
  <Weight>Weight: 75.5</Weight>
</R>
</T>
</P>

```

Figure 4.8 PO.xml

4.3 Displaying the files

To present the result with a nice layout in a web-browser we decided to use HTML and divide the page into two frames. In the left frame a list of the generated classes will appear. By clicking the links, the corresponding information about that class will appear in the right frame.

The list with the links, which is a HTML-file, is amended for each hit (Fig. 4.9).

```

<HTML>
<HEAD>
<TITLE>LINKS</TITLE>
</HEAD>
<BODY>
<A HREF="backUp.xml" TARGET="CONTENT">Info</A><BR>
</BODY>
<A HREF="P1.xml" TARGET="CONTENT">Subscriber 06-123 4567</A><BR>
<A HREF="P2.xml" TARGET="CONTENT">Subscriber 07-123 4567</A><BR>
<A HREF="P3.xml" TARGET="CONTENT">Subscriber 08-123 4567</A><BR>
</HTML>

```

Figure 4.9 list.html

4.4 Running the application

The TBSE application can either be run in a non-graphical or a graphical version. The non-graphical version is run in the command window. The graphical version (figure 4.10) is a GUI-implementation using the swing-package in Java. The GUI includes several options that are described in detail in the TBSE Users Guide (Appendix A).

4.4.1 Test-run (Searching for a class)

In this test-run we are interested of all classes with the name “Subscriber” independent of their keys. So by leaving the text-field for key empty the application starts searching through the backup files for the wanted class name.

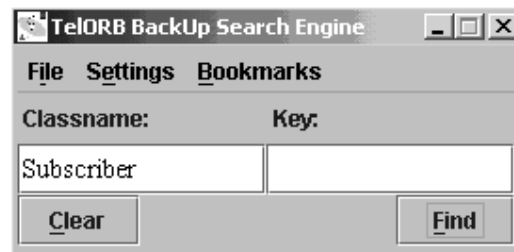


Figure 4.10 TBSE GUI

The application prints out information about what actually is happening at runtime (figure 4.11). For example, in this test-run the application opens the file `lm.1.0.0.ddbe` and looks for “Subscriber”. If no matches were found a message would have showed up. But in this case “Subscriber” is found and then the application starts searching for the related file `lm.1003000.0.0.ddbe` that contains the information to “Subscriber”. While the application is reading through this file it will generate the XML, DTD and CSS files.

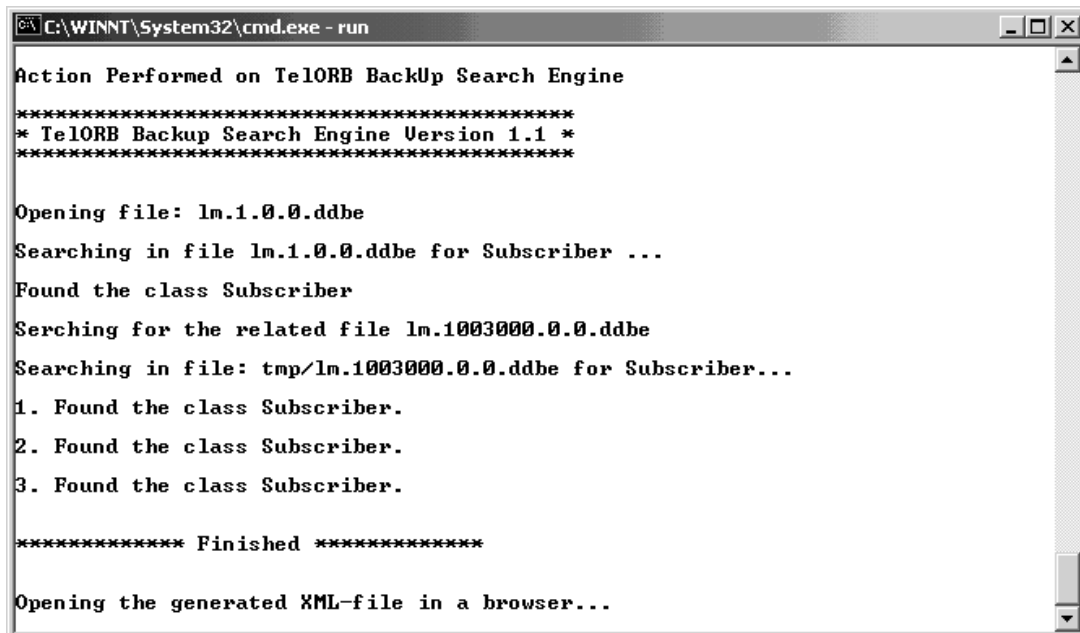


Figure 4.11 Debug information

When the execution process is finished, the generated files are opened in a browser (figure 4.12). As seen the application found three classes with the name “Subscriber” and since no specific key was given, all three classes are displayed. By clicking on the links on the left column the information about the chosen class will be displayed. The first link “info” contains the header information for the classes.



Figure 4.12 The generated XML-file in MS-Explorer

4.4.1 Test-run (Searching for a class with a specific key)

Here we are interested to find the class by the name “Subscriber” with a specific key (06-123 4567), which in this case is the phone number of the subscriber.

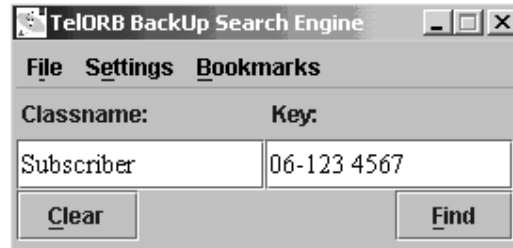


Figure 4.13 TBSE GUI

As seen in the debug information below (figure 4.14) the application could find only one class with the right name and the right key.

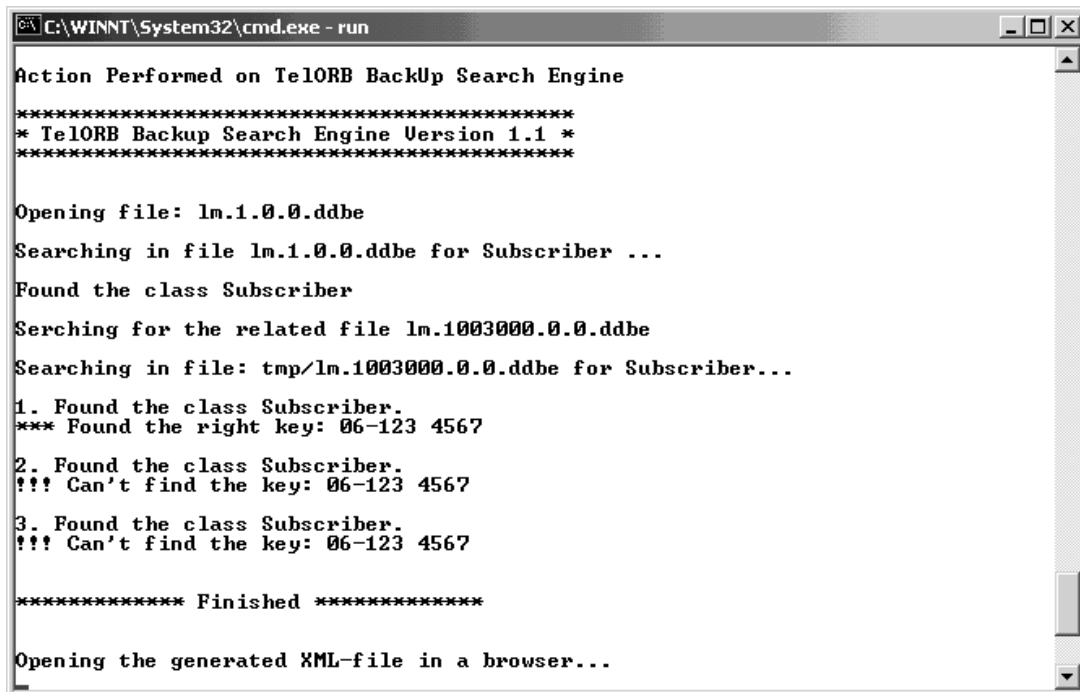


Figure 4.14 Debug information

Only XML, DTD and CSS files for the class with the key “06-123 4567” will be generated and displayed in the browser (figure 4.15).

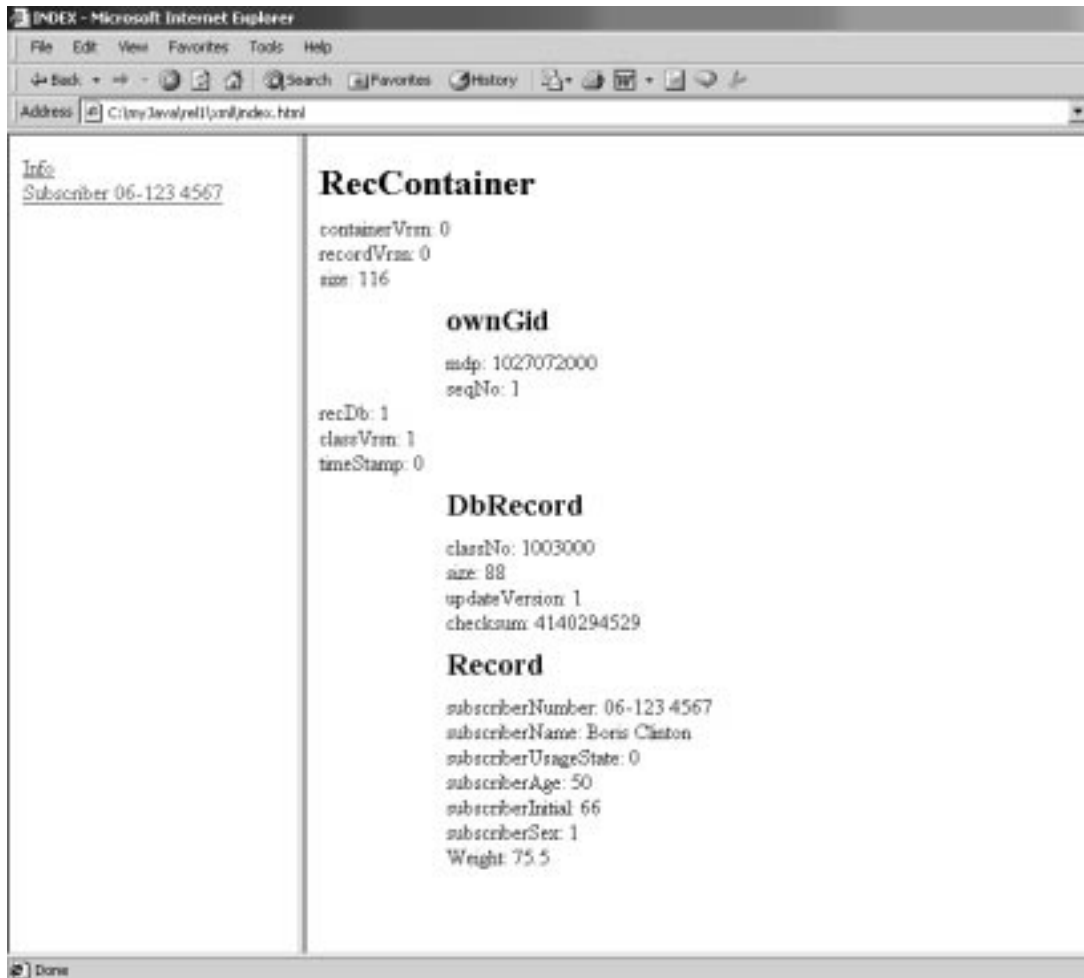


Figure 4.15 The class “Subscriber” with the key “06-123 4567”

Conclusions and future work

This thesis has presented two things, XML and how XML can be used with TelORB.

Since XML gives us a single, human-readable syntax for serializing just about any kind of structured data, in a way that lets it be manipulated and displayed using simple standardized tools, it can be used in almost any kind of modern software development. XML can do for data what Java has done for programs, which is to make the data both platform- and vendor-independent. The only thing that currently is holding back a more widespread public usage of XML is the lack of tools that support XML, but hopefully these tools will be available in the nearest future.

As we have seen it is possible to search after a certain class and even a certain class with a specific key in TelORBs database backup files. With the application we have developed, it is possible to generate XML-documents with corresponding DTD- and CSS-files. This means that we can export a database object from the flat structure in the backup-file to a more structured and understandable format that could be presented in a browser as an XML-document.

Now when we become familiar with TelORB we think that it should be very interesting from TelORBs point of view to import an object to the TelORB database from an XML-document (Fig 5.1).

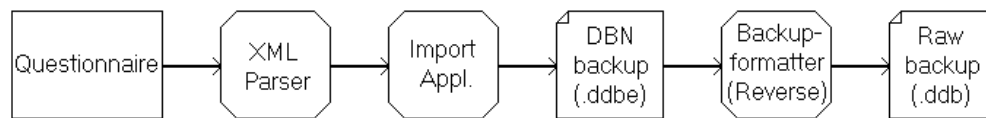


Figure 5.1 Import schematics

The idea is that a questionnaire is implemented where the user fills in the information that should be imported. By clicking on the "send button" the application will take care of the rest. With this tool it should be easy to add new objects to the database or edit existing objects.

We have examined how this can be preformed. It is a reverse process of the export process with some small changes. XML-Schemas should be used instead for a DTD to control the data-types of the input, so it would not be possible to enter a telephone-number in the field earmarked for the name. Just like in the export process where we wrote our own parser to check the input-file for faults, a parser could be used for import as well. There is no reason to develop this parser because there are several

parsers with this function on the market to choose from. A problem is that currently a reverse version of the Backup-formatter does not exist. This limits the import-process so that an object can not be imported all the way into the database. However, there are plans to develop a "reverse-backup-formatter" in the near future, and ones it is available our application can be developed further and used with TelORB.

Dictionary

CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheet
DBN	TelORB database
DID	Database Identifier
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
GNU	GNU's Not UNIX
GUI	Graphical User Interface
HTML	Hypertext Mark-up Language
I/O	Input/Output
IDL	Interface Definition Language
IIOB	Internet Inter-ORB Protocol
IP	Internet Protocol
IPC	Inter Process Communication
ISO	International Standards Organisation
JVM	Java Virtual Machine
POT	Persistent Object Type
SGML	Standard Generalised Mark-up Language
TBSE	TelORB Backup Search Engine
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access

URI	Uniform Resource Identifier
URL	Uniform Resource Locators
W3C	World Wide Web Consortium
Xlink	XML Link Language
XML	eXtensible Mark-up Language
Xpointer	XML Pointer Language
XSL	Extensible Style Language

References

- [1] **TelORB Homepage (Intranet)**
<http://telorb.uab.ericsson.se/>
- Ericsson Review No. 3, 99**
<http://www.ericsson.se/review/pdf/1999037.pdf>
- TelORB Software Implementation**
Selfstudy material
February 2000
- [2] **Java Virtual Machine**
Jon Meyer & Troy Downing
ISBN 1-56592-194-1
- [3] **CORBA SECURITY**
Bob Blakley
ISBN 0-201-32565-9
- [4] **XML-Bible**
Elliotte Rusty Harold
ISBN 0-7645-3236-7
- XML: Extensible Markup Language**
Elliotte Rusty Harold
ISBN 0-7645-3199-9
- A study of XML**
Pontus Norman
February 1999
- XML-Guiden**
Magnus Eriksson
<http://www.acc.umu.se/~alpha/xml/main.html>
- XML-Forum**
Edvina AB, Sollentuna
<http://www.xml-forum.com>
- [5] **Overview of SGML Resources**
<http://www.w3.org/MarkUp/SGML>

- [6] **W3C**
The World Wide Web Consortium
<http://www.w3.org>
- [7] **HTML 4.01 Specification**
<http://www.w3.org/TR/REC-html40>
- [8] **INTERNATIONAL ORGANISATION
FOR STANDARDIZATION Homepage**
<http://www.iso.ch>
- [9] **Advanced Techniques for Java Developers**
Daniel J. Berg
ISBN 0-471-32718-2
- Java direkt**
Jan Skansholm
ISBN 91-44-01244-6
- Java™ 2 Platform**
API Specification
<http://java.sun.com/products/jdk/1.2/docs/api>
- JavaSoft Homepage**
<http://www.javasoft.com>
- The Java™ Tutorial**
A practical guide for programmers
<http://java.sun.com/docs/books/tutorial>
- [10] **The IT-Specific encyclopaedia**
<http://www.whatis.com>
- [11] **The C++ Programming Language**
<http://akpublic.research.att.com/~bs/C++.html>
- [12] **DocZilla**
SGML/XML Module
<http://www.doczilla.com>
- [13] **Netscape Netcenter**
<http://www.netscape.com>
- [14] **Mozilla**
An open-source XML browser
<http://www.mozilla.com>

- [15] **Microsoft Homepage**
<http://www.microsoft.com>
- [16] **Linux Online**
<http://www.linux.org>
- [17] **Sun Microsystems Homepage**
<http://www.sun.com>
- [18] **XML Spy**
The first true IDE for XML
<http://new.xmlspy.com>
- [19] **UNIX-System**
<http://www.unix-systems.org>
- [20] **JavaCC Documentation**
<http://java.cs.uni-magdeburg.de/dokus/JavaCC>

TelORB Backup Search Engine (TBSE)

User's Guide

Requirements

To run the TelORB Backup Search Engine you need jdk 1.2 or later. However the GUI requires that the swing package is available. The PATH variable should be set if you want to be able to conveniently run the application from any directory without having to type the full path of the command. If you don't set the PATH variable, you need to specify the full path to the jdk1.2\bin directory every time you run it, such as:

C:\jdk1.2\bin\java GUI

It's useful to set the PATH permanently so it will persist after rebooting.

Windows NT/2000

This application is configured to run on a Windows platform since the results automatically will be displayed in a MS explorer. However you should add the Internet Explorer to the PATH variable to make this work. Set the PATH as follows:

Start the Control Panel, select System, select Environment, and look for "Path" in the User Variables and System Variables. If you're not sure where to add the path, add it to the right end of the "Path" in the User Variables. A typical value for PATH is:

C:\Program Files\Internet Explorer

Capitalization doesn't matter. Click "Set", "OK" or "Apply".

When this is done start unpacking the TBSE.zip file to the desired directory where you will get a subdirectory called TBSE and all the necessary files. This requires e.g. WinZip, which you can download from the WinZip homepage for free.

To run the graphical version of this application change to the TBSE directory and double click on the *run* bat-file or type *run* in a command-window. If you feel more comfortable using the non-graphical version type double click on the *tbse* bat-file or type *tbse* in a command-window.

Linux or Solaris

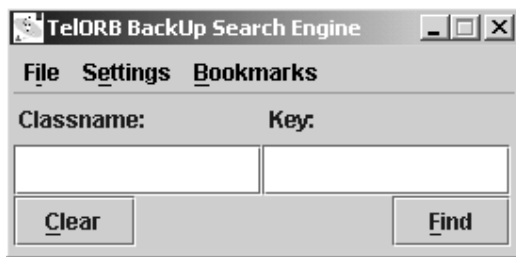
If you run this application on a Linux or Solaris platform you will have to open an XML-browser e.g. Mozilla manually to be able to view the results.

Start with unpacking the TBSE.tar file to the desired directory where you will get a subdirectory called TBSE and all the necessary files. To do this you can type the command *tar xvf TBSE.tar* in a shell.

To run the graphical version of this application, change to the TBSE directory and type *java GUI* in a shell. If you feel more comfortable using the non-graphical version just type *java tbse*.

Using the TelORB Backup Search Engine

When you run the application the following window will appear:



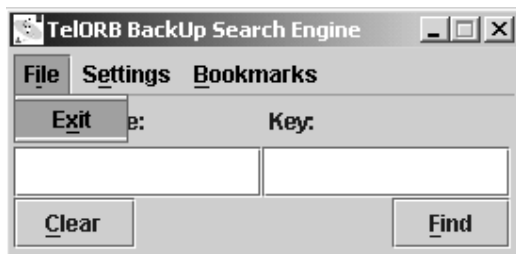
Description of the window elements:

- File: Gives the opportunity to exit the application.
- Settings: Gives the opportunity to set font size, colour, etc.
- Bookmarks: Gives the opportunity to make bookmarks.
- Classname: The field to type the sought class-name.
- Key: The field to type the sought key.
- Clear: Clears the search fields.
- Find: Starts the search.

Notice that you can use short commandos (ALT + the underlined letters) to access the menus without using the mouse.

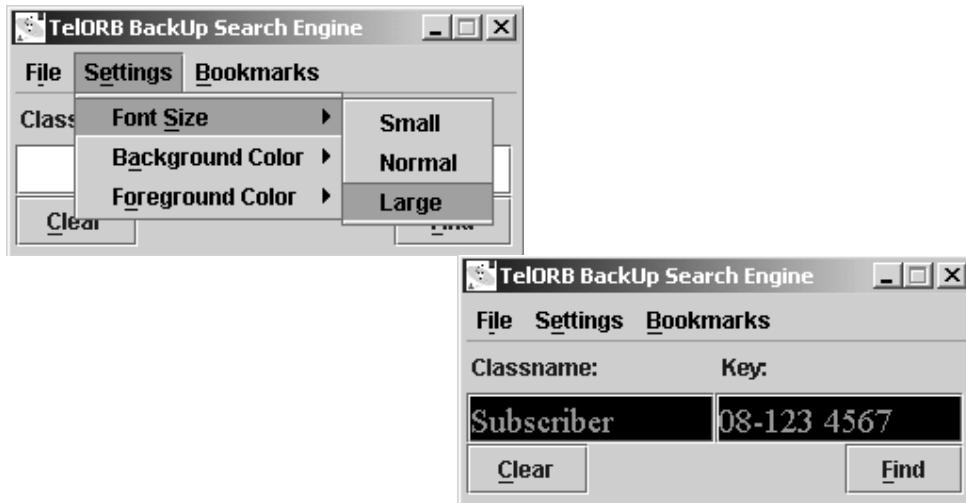
File:

Choosing Exit in this option will terminate the window and likewise the application will stop running.



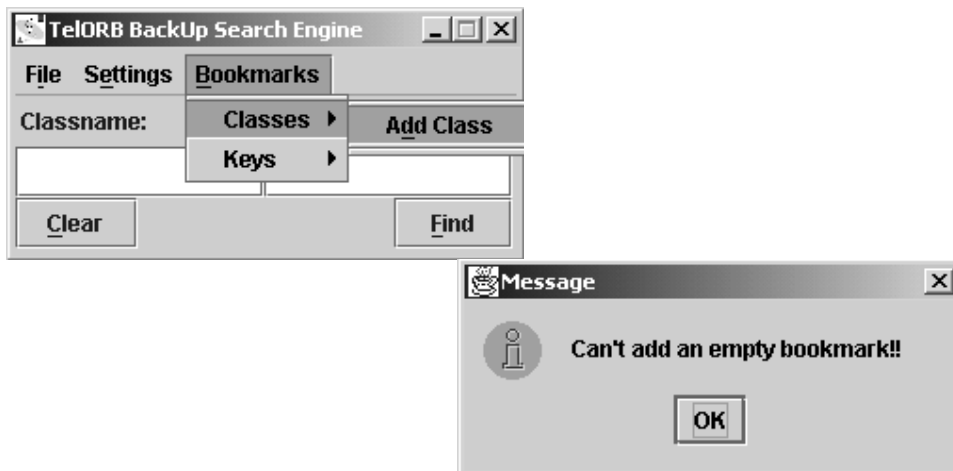
Settings:

Settings give you the option to change the font to a smaller or a larger size. The font size is set to Normal as default. You are also able to change the background and the foreground colour in the text fields.



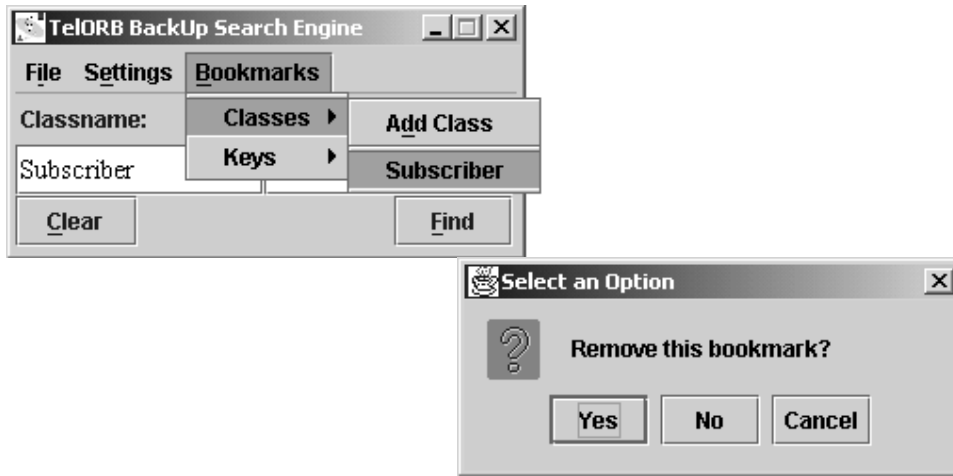
Bookmarks:

You can add the sought class name or/and key to bookmarks by choosing Add Class and/or Add Key. If you try to add a bookmark while the text field is empty a message box will pop up as shown below.



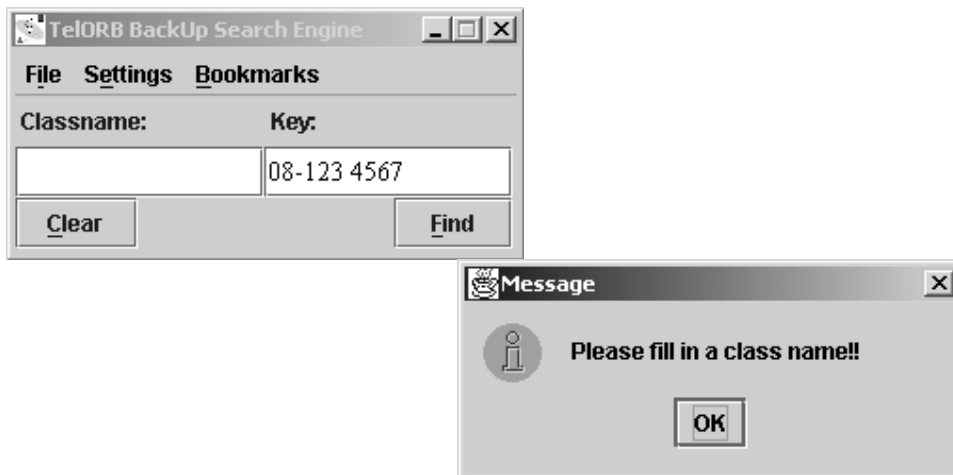
When you want to use a bookmark simply choose the bookmark you want by clicking on it and it will automatically appear in one of the text fields depending on whether it was a classname or a key. This way you don't have to rewrite the class name or/and key that are frequently searched for.

You can also remove a bookmark by clicking on it with the right mouse button. A confirm window will appear as shown below making sure that you really want to delete the bookmark. Choose Yes, No or Cancel depending on what you want to do.

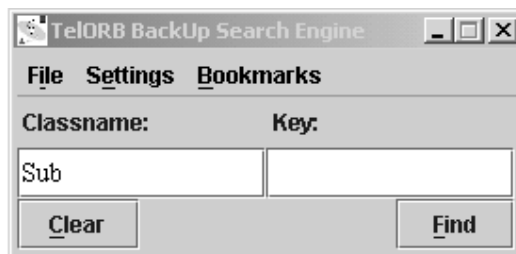


Classname:

This is where you type the class name that you want to find. To start the search hit the Find button. Using the Clear button will clear the text fields. Every time you start a search with or without the key you will have to fill in a class name, otherwise a message box will pop up, as shown below.

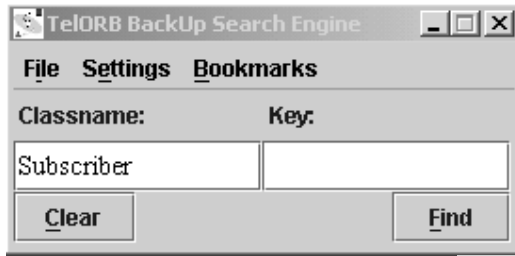


Whenever the sought class name can't be find in the database a message box will pop up as shown below.



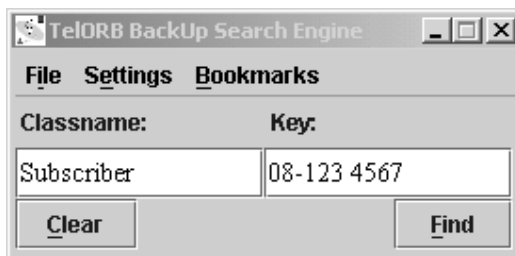


If there were a parse exception in any of the files that the application searched in a message box will pop up as shown below indicating that.



Key:

You can search for a specific key in a class by typing the key in the right text field. Notice that you still have to fill in a class name in which the application will look for the sought key. If any of the related files to either the sought class or the sought key is missing a Message box will pop up and shows which file is missing.



Whenever the application finds the right class and possibly the right key some XML document will be generated which can be viewed in an XML-browser. As mentioned earlier if you run this application in NT, MS explorer will open the resulting XML-files automatically, but in Linux and Solaris you will have to open them manually.

Appendix B

DBN External Format Guide

This document describes both the data that should be included in the TelORB backups and the syntax for it. Also a simple example is given.

Besides some header information the following is included in the backup:

- The meta class record that describes the class record,
- The object schemas, i.e. the class records,
- The instance records, i.e. the records described by the class records.

The syntax and the record part will be described first because they are most interesting from the OaM (Operation and Maintenance) point of view.

The Syntax

The syntax is choosed to be based on context free ASCII characters, to make it easy to generate, scan and fetch selected data with suitable UNIX commands such as 'grep'. It's also preferable that the syntax is compact because lots of data will be transported. Other types of syntaxes, such as XML could be considered later on.

The main point with the syntax is:

- 1) To serve as input for further processing to extract data to the OaM database.
- 2) To be a basis for bulk transfer of data for net redundancy.
- 3) To make an easy manageable dump of the database for debugging purposes
- 4) To be used for an extreme system upgrade or a catastrophic patching.

Therefore all essential information in the backup must be included, though not everything will be needed in the OaM database.

The formats

Primitive values:

$T(v)$ where T is a type tag and v is the value.

Composed values:

$T[:V:V:V:V]$ where T is a type tag and V is a value that can be either primitive or composed.

The list of the values in a composed value may (but need not) be prefixed by a comment, such as an attribute name, before the colon.

A commented value looks like:

comment:V

With the exception of ':' all the other characters are allowed in the comments. It's prudent to avoid '(', ')', 'I' and 'J', though.

Following primitive types exist:

Integer: *(snnn)*
Unsigned: *U(nnn)*
Double: *D(sx.yyyEzz)*
Char: *C(x)* or *C(0xnn)* or *C(\nnn)*
Bool: *B(0)* or *B(1)*
Octet: *O(nnn)* or *O(0xnn)*
String: *S<size>(xxxxxxxxxxxxxxxx)*

How to use the String type is not clear at the moment. The DelosString is allocated as an opaque type in DBN. The syntax for the opaque type will be defined later, at latest in the release at the end of August. Until then the opaque types are exported as octet arrays.

The syntax for the composed values:

Gid

Short form: **G[:U(nnn):U(nnn)]**

Long form: G[mdp:U(nnn)seqNo:U(nnn)]

Array

Short form: A<size>[:V:V:V:V ..:V]

e.g. A4[:V:V:V:V]

Long form: A<size>[0:V 1:V 2:V 3:V ..N:V]

e.g. A4[0:V 1:V 2:V 3:V]

The size is given for both static and dynamic arrays.

Multiref

M<size>[:G[~]:G[~] ..:G[~]]

Record

Short form: R[:V:V:V:V ..:V]

Long form: R[Name1:V Name2:V Name3:V Name4:V ..NameN:V]

DbRecord

Short form: T[:U(nnn):U(nnn):U(nnn):U(nnn):R[~]]

Long form: T[classNo:U(nnn)size:U(nnn)updateVrsn:U(nnn)
checksum:U(nnn)theRecord:R[~]]

RecContainer

Short form: P[:U(nnn):U(nnn):U(nnn):G[~]:U(nnn):
U(nnn):U(nnn):T[~]]

Long form: P[containerVrsn:U(nnn)recordVrsn:U(nnn)
size:U(nnn)ownGid:G[~]recDb:U(nnn)classVrsn:U(
nnn)timeStamp:U(nnn)dbRecord:T[~]]

An Example

Delos-specification of an example POT:

```
OBJECT TYPE AlarmSubscriptionPOT
PROPERTIES
PERSISTENT
    PRIMARY KEY eventType;
    DISTRIBUTED AS EventSubscriptionDUT;

IS
ATTRIBUTES
    eventType           : DelosString;
    consumerNr         : DelosUnsigned;
    sendClearNotification : DelosBoolean;
    subscribers        : REFERENCE TO MANY
AlarmSubscriberPOT;
END;
```

The export format for an instance of AlarmSubscriptionPOT will look like this:

```
T[classNo:U(0x0F43EB)size:U(0x28)updateVrsn:U(0x00)
checksum:U(0x00)theRecord:R[eventType:A5[0:O('A')
1:O('L') 2:O('A') 3:O('R') 4:O('M')]consumerNr:U
(0x01)sendClearNotification:B(0) subscribers:M2[0:
G[mdp:U(0x3D0FA800)seqNo:U(0x01)] 1:G[mdp:U(0x3D0FA800)
seqNo:U(0x02)]]]]
```

The record instance

I) Header information

```
containerVrsn    : unsigned32
recordVrsn      : unsigned32
size            : unsigned32 (header + record)
ownGid          : (unsigned32, unsigned32)
classMdp        : unsigned32
classVrsn       : unsigned32
timeStamp       : unsigned32
```

II) The record

```
classNo      : unsigned32
recordSize   : unsigned32
updateVrsn   : unsigned32
checksum     : unsigned32
```

The meta class

The meta class and the class record data as denoted below will be used for DBN internal purposes. An extract of them will be generated as a redundant schema file to be used by the OaM database.

I) Header information

```
containerVrsn : unsigned32
recordVrsn    : unsigned32
size          : unsigned32 (header + record)
ownGid        : (unsigned32, unsigned32)
classMdp      : unsigned32
classVrsn     : unsigned32
timeStamp     : unsigned32
```

II) The record

```
classNo      : unsigned32
recordSize   : unsigned32
updateVrsn   : unsigned32
checksum     : unsigned32
databaseNo   : unsigned32
classNo      : unsigned32
recordVrsn   : unsigned32
property     : unsigned32
primaryKeySliceNo : unsigned32
primaryKeyAttrNo : unsigned32
mdpPhysBaseNo : unsigned32
noOfMdps    : unsigned32
synchStaticRecordSize : unsigned32
asynchStaticRecordSize : unsigned32
className    : dynamic opaque
classHierarchy : dynamic array of
                unsigned32
attrNoBaseInSlice : dynamic array of
                unsigned32
attrInfo     : dynamic array of attrInfo
                //see below
synchDfltRecord : dynamic opaque
asynchDflts   : dynamic array of dynamic
                opaque
synchStaticAttrNos : dynamic array of unsigned32
synchDynAttrNos   : dynamic array of unsigned32
asynchStaticAttrNos : dynamic array of unsigned32
asynchDynAttrNos   : dynamic array of unsigned32
counterAttrNos     : dynamic array of unsigned32
dbnToolInfo        : dynamic opaque
```

III) AttrInfo

```
type           : unsigned32 //see types below
offset         : unsigned32
size          : unsigned32
extra1        : unsigned32(union{noOfElems,struct
                        {attrNo,sliceNo}})
extra2        : unsigned32(union{classNo,startBit,
                        dflt, asyncAttrNo})
```

The class record

Also for DBN internal purposes only.

I) Header information

```
containerVrsn : unsigned32
recordVrsn    : unsigned32
size          : unsigned32 (header + record)
ownGid       : (unsigned32, unsigned32)
classMdp     : unsigned32
classVrsn    : unsigned32
timeStamp    : unsigned32
```

II) The record

Same as for record instance

For each attribute

```
attributeType : unsigned32
attributeNo   : unsigned32 (starting from 0) or
attributeName : dyn opaque
```

For static attributes

```
attributeValue
```

For static array with static elements

```
noOfElements : unsigned32
elementValue * noOfElements
```

For static array with dynamic elements

```
noOfElements : unsigned32
(elementSize : unsigned32
elementValue) * noOfElements
```

For reference attribute

```
referenceGid : (unsigned32, unsigned32)
```

For dynamic attributes

For dynamicOpaque

```
size           : unsigned32
attributeValue : bit string
```

For multiReference

```
noOfElements   : unsigned32
referenceGid    : (unsigned32, unsigned32) *
noOfElements
```

For dynamicArray with static elements

```
noOfElements   : unsigned32
elementType     : unsigned32
elementValue * noOfElements
```

For dynamicArray with dynamic elements

```
noOfElements   : unsigned32
elementType     : unsigned32
(elementSize   : unsigned32
elementValue) * noOfElements
```

There are special type codes for the different attribute types. These are enumerated below.

Static attribute types

```
char8           = 0x1,
unsignedChar8   = 0x2,
short16         = 0x3,
unsignedShort16 = 0x4,
int32           = 0x5,
unsignedInt32   = 0x6,
long64          = 0x7,
unsignedLong64  = 0x8,
float32         = 0x9,
float64         = 0xa,

reference        = 0x40,
statOpaque      = 0x80,

bit             = 0x200,
bcd             = 0x400,

counterGauge    = 0x1000,
counterPeg      = 0x2000,
conterAcc       = 0x3000

staticArray     = 0x8000
```


Dynamic attribute types

```
dynamicOpaque    = 0x40000
multiReference   = 0x100040
dynamicArray     = 0x400000
```

The meta class expressed according the syntax

(New lines are there for readability reasons...)

```
P[
  containerVrsn:U(0)
  recordVrsn:U(1)
  size:U(918)
  ownGid:G[mdp:U(1)
  seqNo:(1)]
  classMdp:U(1)
  classVrsn:U(1)
  timeStamp:U(1200)
  dbRecord:
    T[
      classNo:U(1)
      recordSize:U(882)
      updateVrsn:U(1)
      checksum:U(12345)
      theRecord:
        R[
          //static part
          databaseNo:U(1)
          classNo:U(1)
          recordVrsn:U(1)
          property:U(49)
          primKeySliceNo:U(0)
          primKeyAttrNo:U(1)
          mdpPhysBaseNo:U(0)
          noOfMdps:U(1)
          synchStaticRecordSize:U(152)
          asynchStaticRecordSize:U(0)
          //dynamic part
          className:
            A12[
              0:O('D') 1:O('i') 2:O('c') 3:O('o') 4:O('s')
              5:O('D') 6:O('b') 7:O('C') 8:O('l') 9:O('a')
              10:O('s') 11:O('s')]
          classHierarchy:
            A1[0:1]
          attrNoBaseInSlice:
            A1[0:0]
          attrInfo:
            A22[
              databaseNo:
                R[
                  type:U(0x06)
                  offset:U(0)
                  size:U(4)
                  extra1:U(0)
                  extra2:U(0)]
              classNo:
                R[
                  type:U(0x06)
```

```

        offset:U(4)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
recordVrsn:
    R[
        type:U(0x06)
        offset:U(8)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
property:
    R[
        type:U(0x06)
        offset:U(12)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
primKeySliceNo:
    R[
        type:U(0x06)
        offset:U(16)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
primKeyAttrNo:
    R[
        type:U(0x06)
        offset:U(20)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
mdpPhysBaseNo:
    R[
        type:U(0x06)
        offset:U(24)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
noOfMdps:
    R[
        type:U(0x06)
        offset:U(28)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
synchStaticRecordSize:
    R[
        type:U(0x06)
        offset:U(32)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
asynchStaticRecordSize:
    R[
        type:U(0x06)
        offset:U(36)
        size:U(4)
        extral:U(0)
        extra2:U(0)]
className:
    R[
        type:U(0x40000)

```

```

        offset:U(40)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
classHierarchy:
    R[
        type:U(0x40000)
        offset:U(48)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
attrNoBaseInSlice:
    R[
        type:U(0x40000)
        offset:U(56)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
attrInfo:
    R[
        type:U(0x40000)
        offset:U(64)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
synchDfltRecord:
    R[
        type:U(0x40000)
        offset:U(72)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
asynchDflts:
    R[
        type:U(0x40000)
        offset:U(80)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
synchStaticAttrNos:
    R[
        type:U(0x40000)
        offset:U(88)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
synchDynAttrNos:
    R[
        type:U(0x40000)
        offset:U(96)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
asynchStaticAttrNos:
    R[
        type:U(0x40000)
        offset:U(104)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
asynchDynAttrNos:
    R[
        type:U(0x40000)

```

```

        offset:U(112)
        size:U(0)
        extral:U(0)
        extra2:U(0)]
counterAttrNos:
  R[
    type:U(0x40000)
    offset:U(120)
    size:U(0)
    extral:U(0)
    extra2:U(0)]
dbnToolInfo:
  R[
    type:U(0x40000)
    offset:U(128)
    size:U(0)
    extral:U(0)
    extra2:U(0)]
  ]
synchDfltRecord:A?[~]
dbnToolInfo:A?[~]
  ]
]
]
]

```