Managing AXE via the Internet

Jörgen Lundberg

Department of Teleinformatics
School of Electrical Engineering and Information Technology
KTH, Royal Institute of Technology

Element Management, UAB/I/MP
Ericsson AXE Research and Development

March 2002

Abstract

The department I/M at Ericsson AXE Research and Development develop software for operation and maintenance of the AXE switching system. I/M have under the project name Formalised MML[1] (FORM) developed a software platform that deals with problems regarding machine-machine interface and man-machine interface.

Until recently, a plain command window in which the AXE operator types commands has been used to operate the AXE switching system. The commands are five letter abbreviations that can have several parameters. The response from the switch (a printout) comes as plain text in ASCII-format and is supposed to be interpreted by a human. Since there are different versions of AXE-switches, there are different versions of printouts for the same command, depending on AXE version. This makes it difficult to make a machine-machine interface that deals with AXE-responses. These two problems were the main reason for the FORM project. The software has been developed in C++ using MOTIF as graphical user interface.

The purpose of this thesis was to investigate different technologies that could make the FORM platform accessible via the World Wide Web and then make a prototype based on one or a combination of these technologies.

One way of making an interactive GUI that is available to a web browser is to develop a Java applet. One of Java's benefits is that it is (in most cases) platform independent.

As mentioned the FORM platform is implemented in C++ and one way of making it possible for programs written in Java to communicate with programmes written in C++ is using CORBA. One of CORBA's benefits is the possibility to distribute different parts of the implementation over a number of machines. In addition, the fact that once the IDL interface is defined you can change the implementation, the implementation language or platform as long as the interface is kept intact.

The prototype that was the second part of this thesis work was developed using Java, OrbixWeb, Orbix and C++ on Solaris 2.5. During testing, the Java applet and CORBA classes for the client were moved to a PC running Windows NT, and with no adjustments, the applet performed exactly as on the Solaris platform.

[1] Man-Machine Language, a set of instructions for communicating with an AXE-switch.

Table of contents

1 Thesis background and introduction

1.1 Managing AXE via the Internet, a master thesis

A Master Thesis marks the end of the Master of Science programme at the Royal Institute of Technology (KTH). A master thesis is a project that spans over 20 weeks and is performed either at a KTH-department or out in the industry. This thesis has had three phases, a literature study, the development of a prototype and finally writing the report.

This thesis is a co-operation between the Department of Teleinformatics at KTH and Ericsson AXE Research and Development (Ericsson Utvecklings AB).

1.2 Problem definition

The initial problem definition was to investigate possible solutions for accessing the FORM platform using Internet technology and designing and implementing a prototype, which uses a web browser as a GUI.

As it turned out there was another thesis project going on at the I/M department with the purpose of designing a Java GUI for a product that uses the FORM platform. That thesis work was done by Lars Dahllöf at the department of Numerical Analysis and Computer Science at KTH.

I decided that there was no use in making another GUI since I had my hands full just trying to understand CORBA. The prototype was therefore developed as a Java application that takes input from a command line.

1.3 The literature study

I started to search for material about HTTP/CGI and Java and it was when reading about Java that I stumbled upon CORBA. Since I have had some experience with HTTP/CGI and Java and basically no experience with CORBA, I dedicated somewhat more time to reading about CORBA.

The literature used was found on the Internet and in the Ericsson (UAB) library. When searching for material on the Internet I started at Sun's Java site[2], at OMG's[3] CORBA site and of course, Digital Equipment's search engine AltaVista[4].

1.4 The prototype

At first, the goal was to make the whole FORM platform available to the Internet. During the prototype development, this turned out

[2] http://www.java.com

[3] http://www.omg.org

[4] http://www.altavista.com

to be a too optimistic goal for a twenty-week thesis. Instead, I chose to concentrate on the FORM platform's parser.

The reason for choosing the parser was because it is (in my opinion) probably the part of the platform that will be used the most. The other parts are mainly help features for the non-expert user. The advanced users will probably use the command line input when sending frequently used commands.

The fact that I did not make the whole FORM platform available to the Internet was no setback for the thesis, since the development procedures for the other parts would have been quite similar to that of the parser. The prototype is described in more detail in chapter 7.

1.5 Limits to the study

The reader of this report is supposed to have knowledge level of a person graduated from the Electrical Engineering programme, having completed an introduction to computer science but not necessarily studied object-oriented programming.

Due to the nature of the World Wide Web, it would have been impossible to investigate all the technologies available. I have tried to keep the chapters on the different technologies rather short and concise in order to keep the report from becoming to wordy. If the reader wants to make deeper studies into a subject, there are a number of books and papers in the references.

Some of the sources come from companies that have a commercial interest in the subject of the article/paper they have written. Example of sources are the papers on Java [10], [11] from Sun Microsystems and [16] from Visigenic. However, these papers do not differ from what seems commonly accepted and consequently it's probably safe to say that the quality of these papers is "high enough".

2 Introduction to the FORM platform

2.1 Ericsson AXE Research and Development

Ericsson AXE Research and Development (UAB) is responsible for the switching system, AXE 10, which has more than 100 million fixed lines in place in some 120 countries.

The AXE system includes:

- telephony switching hardware (APT)
- central processors (CP)
- interfacing regional processors (RP)
- a Support Processor (SP) with the Input/output subsystem (I/O) used in managing and controlling the systems

2.2 The I/O system

The I/O system is the interface between the central processor and whatever systems are used to control and setup the switch. Any changes to the setup, as well as data that needs to be taken out of the central processors, must pass via the I/O system. This includes billing data, statistical data on usage and other parameters, and data used for maintenance, planning and execution. The I/O system is currently developed and maintained at the ‘I’ department at Ericsson AXE Research and Development.

2.3 The FORM platform

It is possible to send commands to an AXE-switch using MML. Sending a command generates an ASCII printout that informs the user about the command sent.

The original MML management interface is in by today’s standards quite primitive[5]. Five letter codes with different parameters must be remembered for a few thousand commands and they are typed directly into a terminal program. The printout is supposed to be interpreted by a human and comes in plain ASCII format.

Formalising MML means introducing formal syntax descriptions of MML-commands and printouts. The syntax descriptions are used by the management tools for operation and maintenance of the AXE10 exchange. This enables generic GUI products to give enhanced support to the operation and maintenance staff. It also supports development of OSS[6] systems with low level parsing thus increasing productivity in OSS development.

[5] The FORM Concepts[7]

[6] Operations Support System

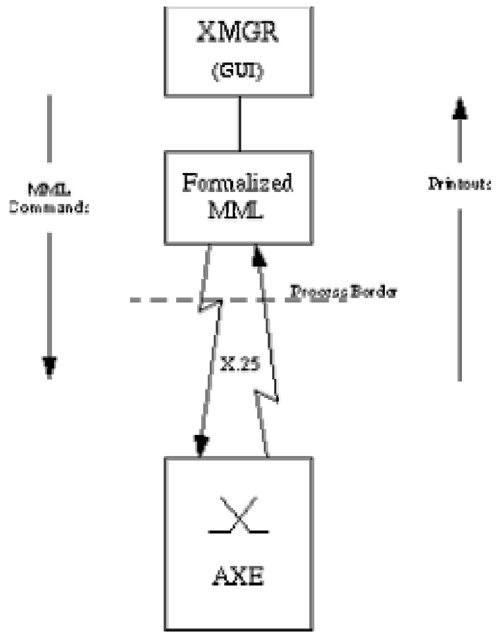

Figure2.1

XMGR is a form-based tool that supports entering of commands. XMGR uses the command syntax information available over an API and produces a form based on that syntax. The form supports the user with information on what parameters there are, which are mandatory/optional and what data types and value ranges the parameters have. XMGR improves the usability of the whole AXE10 user interface and is generic, that is it will not be necessary to upgrade when new commands and new syntaxes are introduced.

The FORM platform consists of different building blocks that communicate with each other and with external components, such as an AXE-switch or a user interface. In Figure 2.2 below, the user interface is the XMGR.

2.4 Using the FORM platform

When sending a command to the switch, it is received by the FORM platform, which in turn passes it on to the switch. When the printout is returned it is passed to the parser by the form platform. Using a POD (Printout Description) the parser parses the printout and sends it back to the FORM platform, which finally sends it back to the application.

The parsing of printouts makes it possible to make machine-machine systems, where the application program isn't a GUI. These applications can respond to alarms and other printouts without any human interaction, and are suitable for routine maintenance type of interaction with the switch.

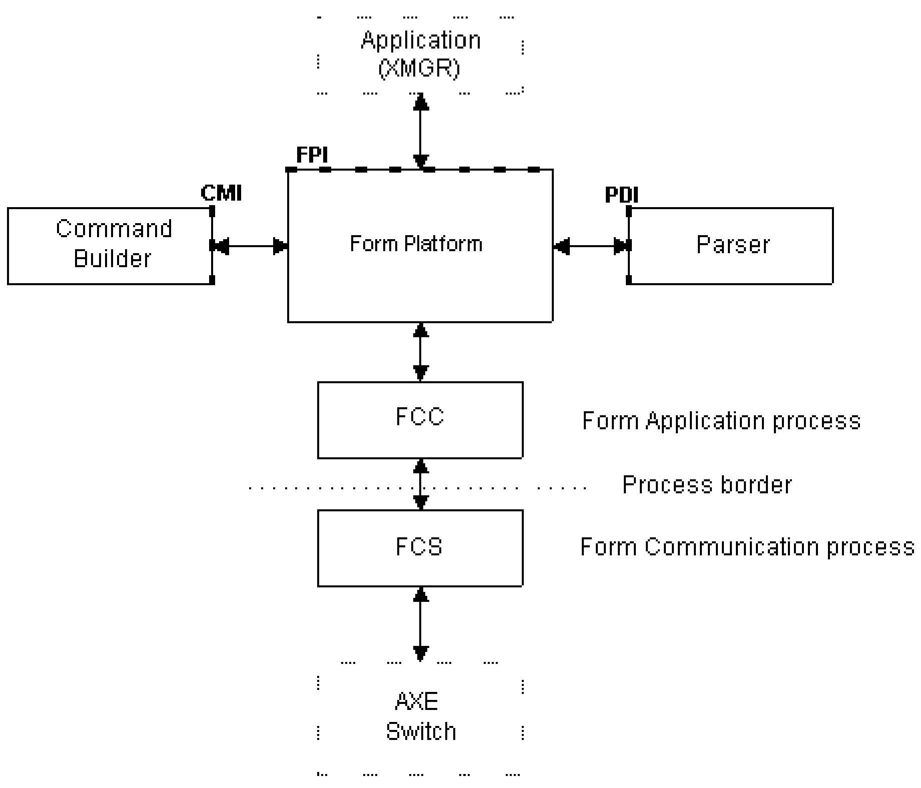

Figure 2.2

Since there was no time to include the whole FORM platform in the prototype, I simulated the switch printouts with text files. This is not too far from reality since the switch printouts come in plain ASCII-format

2.5 The objects of the FORM platform

An application using the FPI (FORM Platform Interface) will have one or more classes that are implementations of the Form Application object, which is a receiver part of the FPI.

The FORM Platform object is the primary object of the platform, and it supplies the station objects and handles registration of an application error handler.

The station object handles connection and disconnection. It supplies the command list and command objects, and the subscription object for spontaneous printouts.

The command object handles the command model, from getting the description from FFI[7], using the Form CMI[8] to build the model, to letting the application access it. It also handles creation/deletion of printout objects.

A printout object receives and contains the printout and handles all tricky bits about printout parsing, using the printout description

[7] Form File Interface XXX

[8] Command Model Interface, see Figure 2.2

interface PDI[9]. It manages immediate, delayed and spontaneous printouts, both parseable and non-parseable.

The subscription object handles the subscription operation. It also supplies printout objects for the spontaneous printouts.

[9] Printout Description Interface

3 WWW, HTML, CGI, …

3.1 Introduction

To a great deal of people, the Internet is synonymous to the WWW; in fact, the WWW is a subset of the Internet. To make it simple, the Internet is a great number of computer networks connected to each other and the WWW is a huge number of files linked to each other on the Internet. These files can contain text, graphics, sound, code snippets and much more. Since the introduction of WWW in the beginning of the nineties, the Internet has grown considerably as shown in Figure 3.1 below

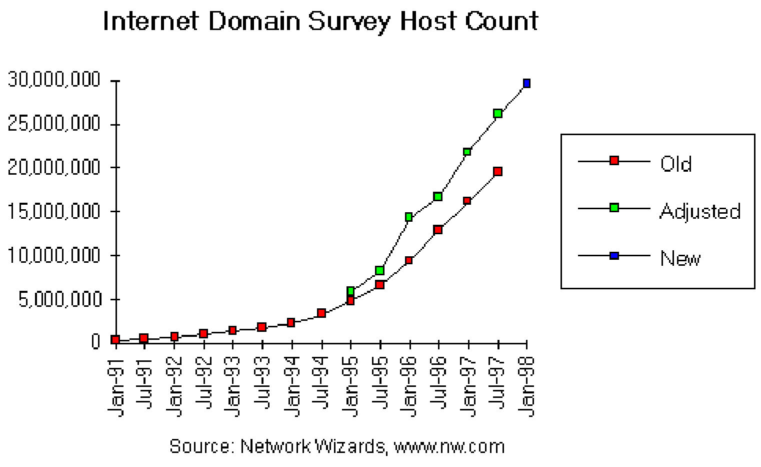

Figure 3.1

3.2 HTML and HTTP

The WWW was invented by Tim Berners-Lee who at the time was working at CERN[10]. He realised that the researchers in high-energy physics were having difficulties in sharing information due to the use of a range of different protocols. On top of that they used a number of different workstation platforms with varying capabilities for displaying graphics and formatted text.

In 1989, Berners-Lee wrote a proposal[11] on how to solve these problems and in 1990, he got the support to start the project.

The basic building blocks of WWW are:

- HTML, Hypertext Markup Language. A layout language for formatting text and graphics.
- HTTP, Hypertext Transfer Protocol. HTTP is based on TCP/IP and makes communication between web browsers and web servers simple.

[10] The European Laboratory for Particle Physics

[11] WorldWideWeb: Proposal for a HyperText Project. [9]

- URL, Universal Resource Locator. A URL is simply a WWW-address pointing out a web resource such as an HTML-file, bitmap or Java applet.

One problem with web resources using only HTML is its limitation in handling user interaction. HTML-pages are static, that is, once they are written they don't change. In order to create an interactive service that can handle user input, an additional technology has to be added. At the moment, there are several such technologies and I will describe a few of them below.

3.3 CGI

3.3.1 Introduction

The abbreviation CGI stands for Common Gateway Interface and is part of HTTP. CGI allow web servers to invoke programs that generate HTML-code "on-the-fly". These programs are often called CGI-bins or CGI-scripts and are accessible via ordinary *s*URLs. The browser displays the HTML-code produced by a CGI-script as if it was a static HTML-page. CGI-scripts make it possible to handle input from the user, connections to databases and so on.

3.3.2 Benefits

- Making a CGI-script is simple, even for the not so experienced programmer.
- CGI-scripts can be implemented in a number of programming languages. One of the more commonly used is Perl[12].

3.3.3 Drawbacks

- Performance. HTTP is quite slow to begin with and on top of that when invoking a CGI-script the web server must start a new process, execute the code and when the script has finished, return the output to the web browser. The performance is dependent of the workload of the server and the performance of any other application invoked by the CGI-script. In addition, while the script is executing, the performance of the web server to other users is lowered.
- CGI-scripts are stateless, meaning that they don't maintain information from one form to the next. The server loses all information as soon as the reply is sent back to the client. One solution to this problem is to write the information to some kind of database, but that will also impair the performance.
- Overhead. Since there is no way of reloading parts of a page, the response from the server comes as a whole HTML-page, although the page might just have changed slightly.
- CGI-scripts can only be used to make basic GUIs. The available GUI-components are the ones provided by HTML-forms, like: buttons, radio buttons, check boxes and selection lists. If a more

[12] Practical and Extendable Report Language

sophisticated GUI is needed, some other technology must be used.

3.4 JavaScript

JavaScript was developed by Netscape and helps in dealing with some of the shortcomings of plain HTML.

JavaScript is an interpreted language that is included in the HTML-code that executes in the web browser (assuming that the browser can interpret JavaScript).

The features provided by JavaScript are similar to that of HTML-forms: buttons, selection lists and the like. In addition, JavaScript can change the appearance of a web page without communicating with a web server. Here are some of the features:

- read browser information
- control browser windows and frames
- image control
- browser cookies
- date and time
- math operations
- detect mouse events
- page history

Below is a small JavaScript example that displays the day of the week. Note that the first five and the last three lines are HTML.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript">
var now=new Date();
var dayNames=new array("Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday");
document.write("Today is "+dayNames[now.getDay()]+.");
</SCRIPT>
</BODY>
</HTML>
```

As shown in the example above, JavaScript is not a substitute for HTML, it’s more of a complement.

JavaScripts execute in the browser, which is an advantage since it doesn’t strain the server. On the other hand, there are different implementations of JavaScript in different browsers, which mean that a JavaScript that works fine in one browser may not wok in another.

3.5 ASP

Active Server Pages was developed by Microsoft and is like CGI a way to produce dynamic HTML-pages. An ASP-page can contain scripts in a number of different languages like VBScript (Visual Basic Script) or JavaScript, and there are third party products that make it possible to use languages like Perl, and REXX.

3.5.1 Benefits

Each ASP-page does not have to run as a separate process on the server, which makes ASP faster than CGI.

3.5.2 Drawbacks

Currently ASP-scripts only run on Microsoft's Windows NT platform.

4 Java

4.1 Introduction

It started in 1990 when Patrick Naughton at SUN Microsystems was leaving SUN for NeXT, SUN gave him an offer he couldn't refuse. He got free hands to create a small group of software designers with one purpose: "To make something cool"[13].

The group was codenamed Green and among other things, they tried to figure out a way for different types of consumer electronics to communicate with each other. Different VCR's, TV's, stereos all had different CPU's, and if a manufacturer wanted to add some feature, they would be limited to what the hardware allows them to do. Furthermore, they have to recompile the source code each time a new processor is introduced. The Green team's thoughts finally lead to a new object-oriented programming language. One of the team members, James Gosling, named it Oak after the tree outside his window.

One of the first applications made with Oak was an interactive remote control with a graphical interface. The remote was called "*7" (star seven). Eventually SUN formed a new company called First Person, based on the Green team. First Person began to develop set-top boxes for video-on-demand. Due to various reasons, the deals with the set-top boxes never came through.

At this time, the WWW was coming to life and the First Person team recommended that they would make Oak available on the Internet. SUN backed the decision to give it away for free. The team wrote an Oak compiler and a web browser with support for running Oak programmes in it called WebRunner. Since Oak was already a registered trademark, Oak was renamed to Java, and WebRunner got the name HotJava. In May 1995 SUN formally announced Java and HotJava at SunWorld '95.

4.2 How does it work?

When compiling a program written in C++ or Pascal the compiler translates the source code directly into machine code.

When compiling a Java program the compiler translates the source code into byte code. The byte code is similar to machine instructions but it is platform independent and can be transmitted over networks to different platforms. In order to run the program the byte code must be fed to a Java Virtual Machine (JVM), which interprets the byte code and transforms it into machine instructions for the specific platform running the program. The Java compiler and the Java Virtual Machine are platform dependent and are available for a number of operating systems.

[13] What is Java? Intro FAQ.
www.javasoft.com[10]

There are two ways of writing Java programs. One way is to write it as a standard stand alone application, the other way is to write it as a program that is meant to be a part of a HTML-page - an applet.

Today a number of web browsers have incorporated a JVM. This makes it possible to download and run Java applets without installing a separate JVM, which is one of the reasons why Java have become so popular in such a short time.

4.3 Advantages

4.3.1 Architecture independent

The JavaSoft's slogan "Write once, run everywhere" is true in most cases and is probably the main reason why Java has become so popular. The Java Developers Kit (JDK) from JavaSoft is available for Solaris and Windows NT/98/95. In addition there is a large number of third party ports available for platforms like MacOs, Linux, Open VMS, Digital UNIX, HP-UX and so on.

On the WWW, there is of course a large number of architectures and the ability of writing code that anyone can download and run on their computer regardless of platform is appealing to most programmers.

The software industry realises that there is money to save if they don't have to make different versions of their software to run on different platforms. There have also been discussions in papers and on the Internet on whether Java will become the technology that breaks Microsoft's dominance on the PC-market. I will leave it to the reader to decide if this is an advantage or not.

4.3.2 Simple and robust

Java has a C++-like syntax and was designed that way since there are many people already familiar with C/C++. However, there are quite a few areas where Java differs from C++. When developing Java the Green team deliberately removed some of the C++ features that in their opinion were unnecessary or error-prone.

> *"Most programmers working these days use C, and most programmers doing object-oriented programming use C++. So even though we found that C++ was unsuitable, we designed Java as closely to C++ as possible to make the system more comprehensible.*
>
> *Java omits many of the rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit"*[14]

Below is a list of some features where Java differs from C++:

- Automatic garbage collection. Programmers don't have to free memory. Instead, unreferenced memory is automatically freed.
- No pointers

[14] The Java Language: An Overview [11]

- Strict object-orientation, i.e. no individual functions or subroutines are allowed.
- No operator overloading.
- No multiple inheritance.
- No structs or unions.

Naturally, there are people who don't agree that removing these features is an advantage. C++ inventor Bjarne Strostrup for instance:

> *"Compared to C++, Java sacrifices efficiency, ability to deal-with low-level facilities, and flexibility of programming for alternatives. ...*
>
> *...I don't think Java is interesting as a programming language. It is an interesting example of mass hysteria and a unique example of a large corporation throwing its marketing clout behind a programming language and marketing it primarily to non-programmers."*[15]

When trying to make things simpler you sometimes have to sacrifice some flexibility. C++ is probably more flexible than Java but, on the other hand, many of the errors made by C++-programmers are related to memory management or misuse of pointers.

The discussions for or against different programming languages often turn in to religious arguments, and with no doubt will the discussions continue.

Java code is checked twice, once during compilation and a second time during runtime. The runtime checking is possible since the Java Virtual Machine interprets the byte code. This feature makes Java robust and it also makes it easier to detect bugs.

4.3.3 Advanced features are built-in.

The Java Developers Kit includes a large number of class libraries. These class libraries, called packages, have a high abstraction level, which means that programmers don't have to deal with low-level programming. Examples of packages are:

- **java.lang** – supplies basic classes like 'String' and 'Boolean', but also 'Threads' that makes multi-threading easy to use.
- **java.net** – supplies classes for easy usage of TCP/IP-sockets, and Internet protocols like HTTP and FTP.
- **java.security** – supplies classes for dealing with security issues.
- **java.rmi** – supplies classes for Java Remote Method Invocation, which is makes it possible to create distributed object-oriented software.

[15] EXE – March 98 [13]

4.4 Disadvantages

4.4.1 Performance

Java byte code is interpreted and translated into machine instructions. This makes it slower than for example C++, which is compiled directly into machine instructions.

In applications where performance is critical, the interpreter is a large disadvantage. Interpreted Java is on the average 20 times slower than compiled C++. However, when running interactive GUI and network applications, where the application is often idle, waiting for user input, the performance is sufficient.

Lately, different software vendors have developed just-in-time (JIT) compilers. A JIT translates performance critical program sections into machine instructions at run-time. Netscape Navigator is one example of an application that incorporates a JIT.

Furthermore, there are traditional source code to machine code compilers available. These compilers make Java programs run almost as fast as C/C++-programs. This however makes the Java programs' architecture dependent and maybe the most important Java feature is lost.

5 Distributed object computing

5.1 Introduction

In order to explain how CORBA works, it is necessary to give an introduction to the main components, namely: object-orientation and distributed computing.

5.2 Object Orientation

Object Orientation (OO) has become one of the leading programming paradigms. An object is a software building block that has an interface for communicating with other objects. The object can contain data and procedures for dealing with the data. These procedures are called methods. There are two main types of methods, public and private. The public methods can be called by other objects to retrieve or manipulate an object's data. The private methods are used inside the object and are hidden to other objects.

This is quite abstract and a bit hard to visualise, but consider the following example:

A bank account with a balance of 500. At a typical ATM, it is possible to retrieve the balance and withdraw some cash. I.e. two methods: "getBalance" and withdraw". There is no way of altering the balance except by using the public methods available.

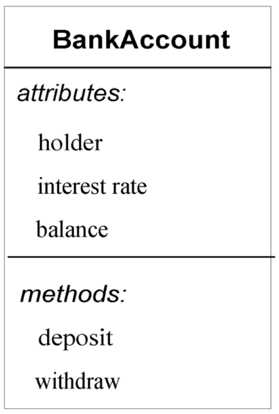

Figure5.1

```
Int  myWallet=100;
BankAccount  myAccount = new BankAccount(500);
myWallet=+myAccount.withdraw(200);
```

In the code above, we have a wallet that contains 100. In line two, an object of type "BankAccount" is created, with 500 as an initial balance. In line three, 200 is withdrawn and put in the wallet. Note that there is a private method, not visible to the user, which alters the balance to 300. However, like in a real ATM this is nothing the user has to deal with.

One important feature in object-oriented design is called inheritance. A bank usually provides a range of different accounts, for example: salary-, savings- and checking accounts. Consider a savings account where you are only allowed to make a certain number of withdrawals each year. If this is the only difference from the bank account used in the example above, we can let the savings account inherit all data items and methods from the bank account. The only thing we need to do is to add the new attribute “max withdrawals”. Inheritance makes it easy to customise existing objects without having to rewrite all of the code.

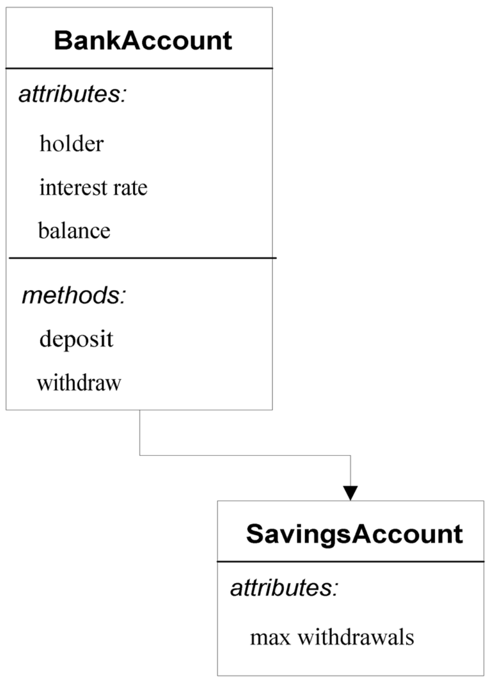

Figure5.2

5.3 Distributed Computing

The word distributed tells us that the data itself and the processing of data is spread out over more than one computer, usually over a network.

In the early days of computing, users were connected to the main computer via “dumb” terminals. As technology moved forward, personal computers, local networks and relational databases became widely used. This led to the birth of the client-server systems.

A client is a program that makes a service request to another program called the server. The client typically contains the presentation logic and some of the business logic. The server is a larger computer that often handles requests from multiple clients. The server often serves as data storage using a database of some sort. As technology evolved further, more complex structures were developed, like multi-layer client-server architectures and systems with multiple nodes, where a node can serve as both a client and a server.

The WWW is the most obvious example of distributed computing. When using a service on the Internet, that service may in its turn access other computers on the WWW to retrieve the requested information.

5.4 Distributed object-orientation

Using distributed object-orientation we get the ability to use features like encapsulation and polymorphism in a distributed environment.

Absolute inter-compatibility between software is the ultimate goal of distributed object-oriented computing. When building a distributed system, developers should ideally be able to add or subtract components without redesigning the system or changing anything else. This component-based system requires further evolution of computing environments and is only possible through sophisticated collaboration mechanisms, including common object services such as global naming, object lifecycle, event filtering, transactional messaging, trade or security services.

Of course, everyone easily understands that standardization of such architecture and services are crucial, and this is where OMG and CORBA steps in.

6 The OMG and CORBA.

6.1 Introduction

CORBA (Common Object Request Broker Architecture) was developed by OMG, the Object Management Group, which is a consortium of software vendors. OMG has over 700 members with giants like SUN Microsystems, Microsoft, IBM and smaller companies like IONA.

The basic idea of CORBA is to provide a platform for distributed object oriented software development.

A CORBA Object Request Broker (ORB) connects a client application with the objects it wants to use. The ORB is like a software bus which connects clients and servers regardless of placement in the network, operation system or implementation language. The client application only needs to keep track of two things: the object's name and how to use the object's interface. The ORB takes care of locating the object, routing the request, and returning the result.

6.2 How CORBA works

A CORBA Object Request Broker (ORB) is the middleware that establishes the client-server relationship between objects. Using an ORB, a client object can invoke a method on a server object that can be on the same machine or across a network. The ORB intercepts the call and finds an object that can implement the request, pass it the parameters, invoke the method, and return the results.

CORBA provides both static and dynamic interfaces to its services. The client does not have to know the object's location, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In addition, the client and server roles are dynamic: an object on the ORB can act as either client or server, depending on the occasion.

Together, ORBs and the CORBA architecture provide the mechanism for CORBA objects to communicate. The objects are small software components that provide some kind of a service, such as access to a database, account management, or inventory tracking.

Fundamental to the architecture are the ORBs. For any client or server to be a part of the CORBA scheme, it must include an ORB to help it find and communicate with other CORBA objects. Once outfitted with an ORB, a client or server can use the services of any CORBA object on any server or host on the network.

CORBA also defines the Internet Inter-ORB Protocol (IIOP) which specifies how ORBs from different vendors interoperate. IIOP is an open protocol that runs on top of TCP/IP. Unlike HTTP, IIOP allows state data to be preserved across multiple invocations of objects and across multiple connections.

6.2.1 The Interface Definition Language, IDL

CORBA defines an interface definition language (IDL) that provides a language-neutral way to describe a CORBA object and the services it provides. IDL lets components written in different languages communicate with each other using IIOP and the rest of the CORBA architecture.

Distributed systems require a number of low-level and repetitious programming efforts, for example:

- Opening, controlling and closing of network connections
- Marshalling and unmarshalling of data (conversion of structured data into programming language independent format and back again)
- Setting up servers to listen for incoming requests on socket ports and forward them to object implementations

IDL compilers and ORB run-time systems free application programmers from these tasks. IDL compilers create representations of IDL-defined constructs such as constants, data types, and interfaces in particular language binding, for example, C++ or Java. They also create the code to marshal and unmarshal the user-defined data types. Libraries are provided to support predefined CORBA types.

The generated code for the client side, that is, the code invoking an operation on an object, is known as stub code. The server side generated code, which invokes the method on the implementation of that operation, is called skeleton code. The skeleton code in conjunction with the ORB provides a transparent run-time mechanism for handling incoming invocations and managing associated network connections.

CORBA objects can reside on different types of systems, including for example Windows, UNIX and DEC VAX mainframes. They can be written in different languages. As long as interfaces to their services are written in IDL, the objects can communicate and use each others' services through ORBs on clients, servers, database systems, mainframes, and other systems on the network.

The IDL-specification for the thesis prototype can be found in Appendix A, and a part of it together with a few explanations in 7.3.

6.2.2 CORBA services

The CORBA architecture provides a set of services that help objects interact with each other. In the CORBA world, services means both the services provided by the CORBA architecture to help objects communicate and the functionality provided by the objects themselves.

The services are among the efficiencies provided by CORBA: You write your object's code, and CORBA takes care of how your object identifies itself, finds other objects, learns of network events, handles object-to-object transactions, and maintains security. The services include the following:

- A *Naming service*, which makes it easy for objects to find another object by name. Each object has a unique name.
- An *Event service*, objects can subscribe to an event channel and be notified of specific events.
- A *Transaction service*, which defines transactional disciplines, co-ordinating two-phase commits among objects.
- A *Security service*, that provides authentication, authorisation, encryption and auditing functions to protect sensitive data and to control user access to applications and services.

6.2.3 BOA

BOA is short for the "Basic Object Adapter". An object adapter is something that supports various styles of object implementations. The BOA was specified first and it supports a multitude of styles in the manner of a Swiss army knife, meaning few implementations ever use all of the services it offers. There are four styles of activation:

- Per Method
- Shared
- Unshared
- Persistent

Per method activation – a new server is started every time an object method is invoked. Each method call runs in its own server.

Shared activation style servers – support multiple objects active at a time. A single server may handle multiple objects all active simultaneously.

Unshared activation servers – only support one single active object. The single server may handle multiple method invocations as long as they are all on the same object.

Persistent servers – are always active and do not require activation. Rather these are presumed to be available as long as the system (or machine) is operating.

6.2.4 Calling a distributed object

An object invocation follows the following call-path through the ORB:

1. The client calls a method through an IDLstub.
2. The ORB hands the request to the BOA, which activates the implementation.
3. The implementation invokes the BOA to inform the BOA that it is active and available.
4. The BOA passes the method request into the implementation via the IDL skeleton.

5. The implementation returns the result (or exception) back to the client through the ORB.

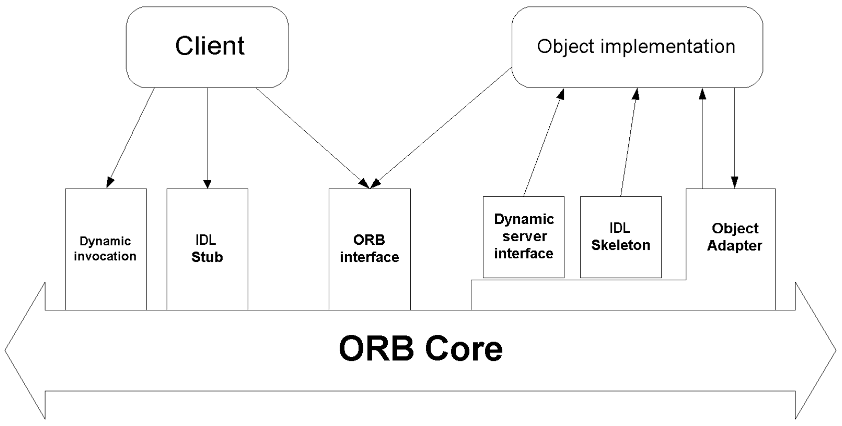

Figure 6.1

6.3 Creating CORBA Objects

These are the basic steps when creating a CORBA object:

1. Write a specification over the objects interface using IDL.
2. Run the specification through an IDL-to-language-compiler. This will result in stubs and skeletons. The stubs are used by the client, and the server skeletons provide the framework that you fill in with the code for the service your object is to provide.
3. Write the code to implement the service.
4. Compile the implementation.

When using Java you don't have to write the specification in the IDL-language. There are Java-IIOP compilers that interpret Java-bytecode and automatically generate the needed stubs and skeletons, which makes it is possible to skip the first two steps in the list above.

6.4 3-tier integration of legacy systems

CORBA enables incorporation of legacy systems into modern client-server systems. A legacy application, module, or entry point can be encapsulated in a C++ or Java "wrapper" that defines an interface to the legacy code. Creating such an object wrapper gives the legacy code a CORBA-compliant interface, making it interoperable with other objects in a distributed computing environment.

On the first tier, there is a client that communicates with a server at the second tier, using IIOP. The server then accesses the legacy system at tier three, also with IIOP. Since CORBA is transparent as far as implementation language and computer architecture is concerned, old systems written in for example Cobol or a database on a mainframe can be accessed.

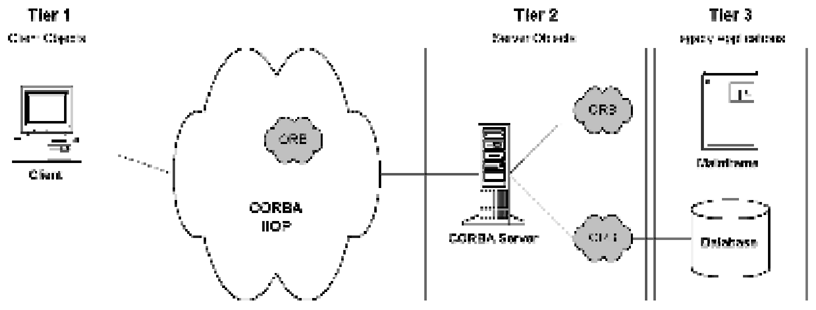

Figure 6.2

In this thesis work, the “legacy system” is the FROM platform.

6.5 Competitive technologies

- HTTP-CGI – The HTTP/CGI is slow, clumsy, and on top of that stateless. This can only be regarded as a good choice in very small applications, but maybe not even then.
- RMI – Remote Method Interface is Java’s distributed object model. Its largest drawback is that it is language dependent. That is, only RMI objects can only talk to other RMI objects and they must be written in Java. RMI does not support interface repositories or dynamic invocations, it does not provide a protocol for secure transactions.
- DCOM – Very few implementations of DCOM run on non-Windows platforms, and for DCOM to scale on the server side, it requires the Microsoft Transaction Server (MTS). MTS is currently only available on Windows NT. In addition, DCOM does not have a distributed naming service, instead it uses the NT registry. DCOM objects do not maintain state between connections. This can be a problem in an environment with faulty connections, like the Internet.
- RPC – Using a Remote Procedure Call you call a specific function (the data is separate). In contrast with an ORB, you call a method within a specific object. Different object classes may respond to the same method call differently. All RPC classes have no specificity: all functions with the same name are implemented the same way.

Below is a table[16] showing performance on remote ping for various client-server techniques. The remote ping example used is perhaps too simple to make any important conclusions, but it gives a hint on the performance of the different methods.

CORBA/IIOP	**DCOM**	**RMI**	**HTTP/CGI**	**Sockets**
3,3 ms	3,9 ms	5,5 ms	603,8 ms	2,0 ms

[16] Instant CORBA, table 2-1, [6]

6.6 Benefits

CORBA is an ISO standard with an open specification and is supported by a large number of hardware and operating system platforms.

CORBA is language independent and thus makes it possible to modify and reuse existing code. In addition, all objects interact via interfaces. The implementation of the interfaces are "hidden" to the application and as long as the interface stays the same developers can modify the object implementation without having to change other parts of the application.

CORBA has location transparency in the sense that objects are located independently of their physical location. Objects may refer to objects that are in the same program, in a different process on the same machine, or located on a remote machine. Each kind of object is used in exactly the same way. Furthermore, an object can potentially change its location without braking the application. The ORB provides the necessary mechanisms for this transparency.

Software designers does not have to worry about low-level programming issues like controlling network connections, marshalling data, and setting up servers to listen for incoming requests. This is all taken cared of by the ORB, client stubs and server skeletons.

Using the Internet Inter-ORB Protocol (IIOP), CORBA objects can interact with other ORBs even if they come from a different software vendor. On top of that, software bridges make it possible for CORBA objects to communicate with Microsoft ActiveX-DCOM objects.

CORBA provides security features like encryption and authentication to protect objects from unauthorised use.

6.7 Drawbacks

CORBA is slower than using sockets, this is natural since it is a middleware. Therefore, if networking performance crucial, you'll probably have to use sockets.

CORBA is quite complex compared to for example HTTP/CGI, and it can take time to understand all of the features.

In order to protect corporate networks from unlawful entries, firewalls between the corporate network and Internet are used. Firewalls make it possible for people on the inside to access www but prevents outsiders to break in (hopefully). This of course makes it harder for ORBs to interact. There are different ways of dealing with this problem, one solution is called HTTP-tunnelling, where the IIOP packets are converted to HTTP packets that the firewall can recognise and deliver to the correct receiver.

7 Accessing the FORM platform via the Internet

7.1 Introduction

I chose to implement the prototype using Java and CORBA mainly for the following reasons:

- Java is platform independent, thus the client program can be used on any computer with a JVM.
- CORBA is an open standard and is also platform independent.
- Using CORBA, it is possible to distribute parts of the FORM platform to different physical machines. This could help to increase performance of the FORM platform.
- It is easy to reuse existing code using a CORBA-“wrapper” which is like a layer surrounding the existing code. This keeps the additional code one has to write to a minimum. This will be explained below.

The scenario for the FORM platform prototype was to reuse “legacy”-code, although “legacy” is a hardly the correct word when talking about the FORM platform. The prototype was made in these steps:

1. Identify the existing interface.
2. Write an IDL-specification of the interface.
3. Run the IDL-specification through an idl-to-java-compiler to retrieve the client stubs, and an idl-to-c++-compiler to retrieve the server skeletons.
4. Write the implementation code for the client and the server.
5. Compile the server skeletons together with the server implementation code to get the “corbyfied” server.
6. Register the server in the CORBA daemon.
7. Compile the client stubs together with the client implementation code to get the “corbyfied” client.

7.2 Identify the existing interface

To simplify the use of the classes in PDI[17], the APDI[18] provides a container class to hide the structure of the PDI, and a set of iterators for retrieving information from a parsed printout. For the prototype, I used the APDI_ParamIterator, which makes it possible to traverse the parse tree a step at the time.

The APDI_ParamIterator class contains the following methods:

- **virtual int next()**
  Steps to the next node in the parse tree.
- **void setParam(const char* theParam)**
  Sets the name to iterate over in the parse tree.

[17] Printout Description Interface

[18] Application based Printout Description Interface

- **virtual const char* getName() const**
  Returns the name of the current node.
- **const char* getType() const**
  Returns the type of the current node.
- **const char* getValue()**
  Returns the value of the current node.
- **void getPosition(int& theRow, int& theColumn) const**
  Returns the position of the current node.

In addition to these methods, the method createParser was added. This method is not specified in APDI_ParamIterator.h. The purpose of createParser is to hide the low level setup of the parse tree and iterator creating.

- **void createParser(in string thePrintOutFile)**

Adding this method was necessary, since I did not distribute the whole FORM-platform. See Appendix C – iter_i.cc for more details.

7.3 Write an IDL-specification of the interface

Now, when we have identified the methods that the client will be able to use, it is time to write an IDL-specification for these methods. The whole specification can be found in Appendix A but I will list a few of the mappings below:

- **long next();**
- **void setParam(in string theParam);**
- **void getPosition(out long row, out long col);**

Operations may have zero or more parameters, and each parameter must be tagged with a keyword to indicate in which direction arguments will be passed at run time.

- **in** – the argument is passed from client to server.
- **out** – the argument is passed from server to client.
- **inout** – the argument is passed from client to server, possibly modified, and then returned.

7.4 Compile the IDL-specification

When using two languages like in the FORM platform prototype, you have to compile the IDL specification twice, once with a IDL-to-Java-compiler and the other with the IDL-C++-compiler.

We now have our client stubs and server skeletons that will take care of the low-level networking issues.

The IDL-to-language compiler also gives us source code skeletons with the method declarations already created. For the server implementation, we get two files named iter_i.h[19] and "iter_i.cc[20]",

[19] Appendix B, iter_i.h

[20] Appendix C, iter_i.cc

where the “i” stands for implementation. Below is a small part of these files to give you an example:

```
//iter_i.h
#ifndef iter_ih
#define iter_ih

class iter_i : public iterBOAImpl
{
public:

   virtual CORBA::Long next(CORBA::Environment &IT_env);

   virtual void setParam(const char * theParam,
                         CORBA::Environment &IT_env);



// iter_i.cc
#include "iter_i.h"
CORBA::Long iter_i::next(CORBA::Environment &IT_env)
{
}

void iter_i::setParam(const char * theParam,
                      CORBA::Environment &IT_env)
{
}
```

All we have to do now is to put our implementation code in each method and add some declarations in the iter_i.h-file.

7.5 Write implementation code for the server object

Since we to create a wrapper for the APDI_ParamIterator we only have to map our methods to the corresponding APDI_ParamIterator methods. In the iter_i.h-file, we have added the line below to give us an APDI_ParamIterator object:

```
    APDI_ParamIterator anIterator;
```

The mapping of the methods in APDI_ParamIterator is now pretty straightforward. For the methods “next” and “setParam” above, we get:

```
CORBA::Long iter_i::next(CORBA::Environment &IT_env)
{
   return anIterator.next();
}

void iter_i::setParam(const char * theParam,
                      CORBA::Environment &IT_env)
{
   anIterator.setParam(theParam);
}
```

To complete the server-side part of this implementation, we also have to write code for the actual server[21]. The server creates a

[21] Appendix D, Srv_Main.cc

server object and tells the ORB that it is ready to receive method calls.

```
// create an iterator object - using
// the implementation class iterator
iter_i *myIterator= new iter_i();

// tell Orbix that we have completed
// the server's initialisation:
CORBA::Orbix.impl_is_ready("iter");
```

7.6 Compile the server skeletons together with the server implementation code to get the "corbyfied" server.

This will give us an executable program that we can register with the CORBA daemon. After that, any method calls will be forwarded by the CORBA daemon to the server object.

7.7 Write implementation code for the client

We can now create our Java client using the methods of APDI_ParamIterator in the same way as we would do with any other non-distributed object. There is however one thing we have to do before we start. We must obtain an object reference for the server object:

```
public static _iterRef myIterObj = null;

// Obtain a reference for the server object
myIterObj = iter._bind(":iter", hostname);
```

As mentioned before, the "createParser" method was added by me to setup the parse tree. Furthermore, to simulate a printout coming from an AXE-switch, I read a printout text file from disk.

```
printoutString=new String(fileToString(printoutFileName));
myIterObj.createParser(printoutString);
```

After these small procedures, we are now able to use the methods of APDI_ParamIterator to traverse the parse tree.

When the client implementation code is written, all we have to do is compile the client stubs together with the client implementation code. This will give us a Java client using objects of the FORM platform.

8 Conclusions

In this thesis work I have described a number of Internet technologies that could be used to make the FORM-platform available to the Internet. Due to obvious reasons, I had to limit the number of technologies in order to fit the necessary work involved, into this thesis project.

When I was about to start implementing the prototype I identified two scenarios:

1. Designing a client with a not so advanced user interface that was not platform- or language independent. This could be done with HTML and a combination of JavaScript and CGI.
2. More freedom when creating the user interface and platform independent client. Using one or more of the technologies that I have described, that would mean using-Java with CORBA as middleware.

I saw greater potential in succeeding with scenario number two, and therefore I choose to go with Java and CORBA.

If you take in to account that prior to this thesis I had only taken two courses in Java and C++ programming and I did not know of CORBA, and that I managed to develop a working prototype, one can assume that it would not be too hard for more experienced developers to create a more advanced CORBA-interface to the whole FORM-platform.

9 References

9.1 Books

9.1.1 C++

[1] Strostrup, Bjarne
The C++ programming language, second edition
1991, Addison Wesley
ISBN: 0-210-53992-6

9.1.2 Perl

[2] Wall, Christiansen, Schwartz
Programming Perl, Second Edition
O'Reilly & Associates, Inc. (1996)
ISBN: 1-56592-1449-6

9.1.3 Java

[3] Flanagan
Java in a Nutshell, Second Edition
O'Reilly & Associates, Inc. (1997)
ISBN: 1-56592-262-X

[4] Vogel, Duddy
Java Programming with CORBA
John Wiley & Sons, Inc. (1997)
ISBN: 0-471-17986-8

9.1.4 CORBA

[5] Baker, Seán
CORBA Distributed Objects Using Orbix
Addison Wesley Longman Ltd
ISBN: 0-201-92475-7

[6] Orfali, Et al.
Instant CORBA
John Wiley & Sons, Inc. (1997)
ISBN:

9.2 Papers and articles

9.2.1 Ericsson Documents

[7] The FORM Concepts, UAB/I/R-96:196 Uen

[8] FPI, FORM Platform Interface Application Programmer's Interface, UAB/I/MT 1997-11-16, 155 10 – CXA 110 214 Uen

9.2.2 WWW

[9] T. Berners-Lee, R. Caillau
WoldWideWeb: Proposal for a HyperText Project, 1990
www.w3c.org/Proposal.html

9.2.3 Java

[10] What is Java? Intro FAQ.
www.javasoft.com

[11] The Java Language: An overview.
www.javasoft.com

[12] Michael O'Connell
Java: The Inside Story.
Sun World Online –July 1995.
www.sun.com/sunworldonline/swol-07-1995/swol-07-java.htm

9.2.4 C++

[13] Will Watts
Bjarne Again. Interview with Bjarne Strostrup.
EXE – March 1998.

9.2.5 CORBA

[14] OrbixWeb Programming Guide
Part of the OrbixWeb documentation

[15] Steve Winoski
CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments.
IEEE Communications Magazine, Vol. 14, No.2, February 1997.

[16] Distributed Object Computing in the Internet Age
Visgenic Software
http://www.visigenic.com/prod/vbrok/wp.html

Appendices

Appendix A. iter.idl

```
// iter.idl
//
// An IDL-interface for methodes in APDI_ParamIterator.h which
// is part of the FORM platform. Check out comments below as well!
//

interface iter {

       void createParser(in string thePrintOutFile);
       // Method for creating the parse tree.
       //
       // This method is NOT specified in APDI_ParamIterator.h!
       // The purpose of createParser is to hide the low level
       // set up of the parse tree and iterator creating.
       //
       // Check out the implementation in "iter_i.cc" and compare
       // with the non-CORBA example in "apdi_test/t_APDI.cc".
       //

       //
       // The methodes below maps methods in APDI_ParamIterator.h
       //

       long next();
       // Steps to the next value in the parse tree.

       void setParam(in string theParam);
       // Sets the name to iterate over in the parse tree

       string getName();
       // Returns name of the current node.

       string getType();
       // Returns type of the current node.

       string getValue();
       // Returns value of the current node.

       void getPosition(out long row, out long col);
       // Returns position of the current node.

};
```

Appendix B. iter_i.h

```
#ifndef iter_ih
#define iter_ih

#include "FMRT_APDI.h"
#include "APDI_ParamIterator.h"
#include "APDI_ParamSetIterator.h"
#include "SCAN_nameFinder.h"
#include "PDI.h"
#include "FMRT_FFI_FileInfo.hh"
#include "iter.hh"

class iter_i : public iterBOAImpl
{
   APDI_ParamIterator anIterator;

public:
   //ctor
   iter_i::iter_i();

   //dtor
   iter_i::~iter_i();

   virtual void createParser(const char * thePrintoutFile,
                             CORBA::Environment &IT_env);

   virtual CORBA::Long next(CORBA::Environment &IT_env);

   virtual void setParam(const char * theParam,
                         CORBA::Environment &IT_env);

   virtual char * getName(CORBA::Environment &IT_env);

   virtual char * getType(CORBA::Environment &IT_env);

   virtual char * getValue(CORBA::Environment &IT_env);

   virtual void getPosition(CORBA::Long& row,
                            CORBA::Long& col,
                            CORBA::Environment &IT_env);
};

#endif
```

Appendix C. iter_i.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strstream.h>
#include <fstream.h>
#include <iostream.h>
#include <sys/param.h>

#include "FMRT_APDI.h"
#include "APDI_ParamIterator.h"
#include "APDI_ParamSetIterator.h"
#include "SCAN_nameFinder.h"
#include "PDI.h"
#include "FMRT_FFI_FileInfo.hh"

#include "iter_i.h"

//ctor
iter_i::iter_i() {}

//dtor
iter_i::~iter_i() {}

void iter_i::createParser (const char* thePrintoutFile,
                           CORBA::Environment &IT_env)
{
   SCAN_nameFinder* myAssoc=SCAN_nameFinder::Instance();
   if (!myAssoc)
   {
      fprintf(stderr , "No SCAN name finder.\n");
      exit(-1);
   }
   PDI::errorReport myError;
   istrstream myInFile(thePrintoutFile, strlen(thePrintoutFile));

   if (!myInFile)
   {
      fprintf(stderr , "Error while creating the istream myInFile\n");
      exit(-1);
   }
   streampos myPos=myInFile.tellg();
   int myPlace=0;
   char myHeader[128];

   myAssoc -> get_header(myInFile , myPlace , myHeader);
   if (myPlace < 0)
   {
      fprintf(stderr , "Cannot fetch the header.\n");
      exit(-1);
   }

   char* myPosFile;
   FMRT_FFI_FileInfo myFileInfo("APZ 211 10 R2, APT 210 08 R5");
   myPosFile=myFileInfo.getPpmFile(myHeader);
   if (!myPosFile)
   {
      fprintf(stderr , "Cannot fetch the header symbol.\n");
      exit(-1);
   }

   FMRT_APDI myApp;
   if (myApp.createParser(myPosFile , myPlace))
   {
      fprintf(stderr , "Unable to create a parser with \"%s\".\n" ,
              myPosFile);
      exit(-1);
   }

   myInFile.seekg(myPos);
   if (myApp.parse(myInFile , myError))
   {
      fprintf(stderr , "Unable to parse.\nError during
              myApp.parse()\n");
      exit(-1);
   }

  myApp.connectIterator(anIterator);
}

CORBA::Long iter_i::next(CORBA::Environment &IT_env)
{
   return anIterator.next();
}

void iter_i::setParam(const char * theParam,
                      CORBA::Environment &IT_env)
{
   anIterator.setParam(theParam);
}

char * iter_i::getName(CORBA::Environment &IT_env)
{
   return CORBA::string_dupl(anIterator.getName());
}

char * iter_i::getType(CORBA::Environment &IT_env)
{
   return CORBA::string_dupl(anIterator.getType());
}

char * iter_i::getValue(CORBA::Environment &IT_env)
{
   return CORBA::string_dupl(anIterator.getValue());
}

void iter_i::getPosition(CORBA::Long& row,
                         CORBA::Long& col,
                         CORBA::Environment &IT_env)
{
   int a, b;
   anIterator.getPosition(a , b);
   row=a;
   col=b;
}
```

Appendix D. Srv_Main.cc

```
#include <stream.h>
#include <stdlib.h>
#include "iter_i.h"

int main()
{
   // create an iterator object - using
   // the implementation class iterator
   iter_i *myIterator= new iter_i();

   try
   {
      // tell Orbix that we have completed the
      // server's initialisation:
      CORBA::Orbix.impl_is_ready("iter");
   }
   catch (CORBA::SystemException &sysEx)
   {
      cerr << "Unexpected system exception" << endl;
      cerr << &sysEx;
      exit(1);
   }
   catch (...)
   {
      // an error occured calling impl_is_ready() - output the error.
      cout << "Unexpected exception" << endl;
      exit(1);
   }

   // impl_is_ready() returns only when Orbix times-out an idle server
   // (or an error occurs).
   cout << "server exiting" << endl;

   return 0;
}
```

Appendix E. iterClient.java

```
package testiter;

import java.io.*;
import IE.Iona.Orbix2.CORBA.IntHolder;
import IE.Iona.Orbix2._CORBA;
import IE.Iona.Orbix2.CORBA.SystemException;

public class iterClient
{
   public static _iterRef myIterObj = null;
   static String printoutString;
   static IntHolder aLine=new IntHolder();
   static IntHolder aCol=new IntHolder();

   public static void main(String args[])
   {
       /**
       * Command line syntax is as follows...
       *
       * iterClient filename paramname
       *
       * filename is a printout ascii file.
       * paramname is a parameter in a printout.
       *
       **/

       boolean useOrbixProt = true;
       String printoutFileName=new String(args[0]);
       String myParam=new String(args[1]);
       String hostname=new String(args[2]);

       if (useOrbixProt)
       {
          // Using the Orbix Protocol
          // So set the Orbix Protocol default values
          //
          // See Configure.java
          _CORBA.IT_BIND_USING_IIOP = false;
       }

       // Obtain a reference for the server object
       try
       {
          myIterObj = iter._bind(":iter", hostname);
       }
       catch (SystemException ex)
       {
          System.out.println("Exception during bind");
          System.out.println(ex.toString());
          System.exit(1);
       }

       // create parser
       try
       {
          printoutString=new String(fileToString(printoutFileName));
       }
       catch (IOException ex)
       {
          System.out.println("FAIL\tException during fileToString");
          System.out.println(ex.toString());
       }
```

```
      try
      {
         myIterObj.createParser(printoutString);
      }
      catch (SystemException ex)
      {
         System.out.println("FAIL\tException during createParser");
         System.out.println(ex.toString());
         return;
      }

      try
      {
         myIterObj.setParam(myParam);
      }
      catch (SystemException ex)
      {
         System.out.println("FAIL\tException during setParam");
         System.out.println(ex.toString());
         return;
      }

      try
      {
         while(myIterObj.next()!=0)
         {
             try
             {
                String aLabel=myIterObj.getValue();
                System.out.println("Its value is "+aLabel);
             }
             catch (SystemException ex)
             {
                System.out.println("FAIL\tException during getValue");
                System.out.println(ex.toString());
                System.exit(-1);
             }
         }
      }
      catch (SystemException ex)
      {
         System.out.println("FAIL\tException during next");
         System.out.println(ex.toString());
         System.exit(-1);
      }
   }

   public static String fileToString(String f) throws IOException
   {
      File inFile=new File(f);
      int fileSize=(int)inFile.length();
      int bytesRead=0;
      FileInputStream inStream=new FileInputStream(inFile);
      byte[] fileData=new byte[fileSize];
      while(bytesRead < fileSize)
      {
         bytesRead+= inStream.read(fileData, bytesRead, fileSize-bytesRead);
      }
      String aString=new String(fileData);
      return aString;
   }
}
```