Data propagation and self-configuring directory services in a distributed environment

Svante Hedin

23 januari 2002
Data propagation and self-configuring directory services in a distributed environment

Svante Hedin

23 januari 2002
Data Propagation and Self-Configuring Directory Services in a Distributed Environment

The Swedish field of digital X-ray imaging has since several years relied heavily on distributed information systems and digital storage containers.

To ensure accurate and safe radiological reporting, Swedish software-firm eCare AB delivers a system called Feedback—the first and only quality assurance IT support product of its kind. This thesis covers several aspects of the design and implementation of future versions of this software platform.

The focus lies on distributed directory services and models for secure and robust data propagation in TCP/IP networks. For data propagation, a new application, InfoBroker, has been designed and implemented to facilitate integration between Feedback and other medical IT support systems. The directory services, introduced in this thesis as the Feedback Directory Services, have been designed on the architectural level. A combination of CORBA and Java Enterprise Edition is suggested as the implementation platform.
Data Propagation and Self-Configuring Directory Services in a Distributed Environment

Master's Thesis in Media Technology and Engineering, Linköping Institute of Technology

Svante Hedin

Norrköping, 23 January 2002
Abstract

The Swedish field of digital x-ray imaging has since several years relied heavily on distributed information systems and digital storage containers.

To ensure accurate and safe radiological reporting, Swedish software-firm eCare AB delivers a system called Feedback — the first and only quality assurance support product of its kind. This thesis covers several aspects of the design and implementation of future versions of this software platform.

The focus lies on distributed directory services and models for secure and robust data propagation in rcr/ir networks. For data propagation, a new application, InfoBroker, has been designed and implemented to facilitate integration between Feedback and other medical rir support systems. The directory services, introduced in this thesis as the Feedback Directory Services, have been designed on the architectural level. A combination of corba and Java Enterprise Edition is suggested as the implementation platform.
About This Thesis

This Master’s thesis is the author’s final thesis at the Master of Science Programme in Media Technology and Engineering, Linköping University, Sweden.

The project is sponsored by eCare AB and all work has been carried out in close cooperation with eCare’s software engineers and market planners.

Targeted Audience

For full appreciation of this thesis, the reader is assumed to have an engineering or computer science background, with some programming skills and a general understanding of computer networks, particularly TCP/IP networks.

That said, large parts of the report are still available for readers without specific technical or mathematical skills.

Work of the Author

- Definition and design of the new concepts Feedback Tree and Feedback Directory Services (FDS).
- Design and implementation of the InfoBroker application.
- Recommendations on implementing FDS and secure and efficient data-propagation mechanisms for binary and textual data.

Acknowledgements

The author wishes to thank the staff at eCare for moral support and encouragement, and in particular Dr. Nina Lundberg, Dr. Johan Hedin, Joachim Wallberg and Dr. Bo Jacobsson for proof-reading, suggestions and many ideas. A word of gratitude also goes to Prof. Björn Kruse at Linköping Institute of Technology for his trust in this project.
# Contents

1 Introduction and Background ................................. 1
   1.1 Background: x-ray Imaging ...................................... 1
   1.2 The Feedback System ........................................... 2
       1.2.1 Overview .............................................. 2
       1.2.2 This Thesis ........................................... 2
       1.2.3 Current Feedback Technical Implementation ............... 3

2 Distributed Computing: an Overview ......................... 7
   2.1 Distributed Computing in a Context ............................ 7
   2.2 General Advantages of Distributed Applications ............... 8
       2.2.1 Fault Tolerance ......................................... 8
       2.2.2 Scalability ............................................. 8
       2.2.3 Minimized Network Load .................................. 8
   2.3 Difficulties with Distributed Applications ................... 8
       2.3.1 Data Integrity .......................................... 8
       2.3.2 Security .............................................. 9
       2.3.3 Complexity ........................................... 9
   2.4 Supporting Distributed Applications — Environments and Protocols . 9
       2.4.1 sql-based Products ...................................... 10
       2.4.2 Remote Procedure Calls and Remote Method Invocation .... 10
       2.4.3 Distributed Transaction Processing Monitors ............. 11
       2.4.4 Object Request Brokers and CORBA ...................... 11
       2.4.5 Message-Oriented Middleware ............................. 15
       2.4.6 Encryption and Authentication ............................ 17
       2.4.7 Choosing Middleware and Appropriate Level of Coupling . 19
       2.4.8 Naming and Directory Services ........................... 20
       2.4.9 Data Propagation ....................................... 23

3 Requirements ................................................... 25
   3.1 Directory Service ............................................... 25
   3.2 Data Propagation ................................................ 26

4 Proposed Solution: Directory Services ....................... 27
   4.1 Introducing the Feedback Tree .................................. 27
   4.2 Comparing Existing Directory Services ......................... 27
       4.2.1 Domain Name System ..................................... 28
       4.2.2 LDAP .................................................. 28
       4.2.3 SMB and NetBIOS ........................................ 29
       4.2.4 CORBA Naming Service .................................. 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>30</td>
</tr>
<tr>
<td>4.3.1</td>
<td>30</td>
</tr>
<tr>
<td>4.3.2</td>
<td>32</td>
</tr>
<tr>
<td>4.3.3</td>
<td>34</td>
</tr>
<tr>
<td>4.3.4</td>
<td>35</td>
</tr>
<tr>
<td>4.3.5</td>
<td>35</td>
</tr>
<tr>
<td>4.3.6</td>
<td>38</td>
</tr>
<tr>
<td>4.3.7</td>
<td>38</td>
</tr>
<tr>
<td>4.3.8</td>
<td>40</td>
</tr>
<tr>
<td>4.3.9</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>45</td>
</tr>
<tr>
<td>5.1.1</td>
<td>46</td>
</tr>
<tr>
<td>5.1.2</td>
<td>47</td>
</tr>
<tr>
<td>5.1.3</td>
<td>48</td>
</tr>
<tr>
<td>5.1.4</td>
<td>49</td>
</tr>
<tr>
<td>5.1.5</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>51</td>
</tr>
<tr>
<td>5.2.1</td>
<td>52</td>
</tr>
<tr>
<td>5.2.2</td>
<td>54</td>
</tr>
<tr>
<td>5.2.3</td>
<td>67</td>
</tr>
<tr>
<td>5.2.4</td>
<td>68</td>
</tr>
<tr>
<td>5.2.5</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>71</td>
</tr>
<tr>
<td>A</td>
<td>73</td>
</tr>
</tbody>
</table>
1.1 Background: x-ray Imaging

x-ray imaging plays an unique role in healthcare. It is the dominant speciality for morphological\(^1\) diagnosis and alternate methods often do not exist. x-ray images are interpreted by humans, who can, of course, make mistakes. Errors can be classified as one of the following:

- Information is overlooked.
- Information is interpreted the wrong way.

The consequences of a misdiagnosis vary depending on many factors, including the patient’s condition, what measures have already been taken, the effects of wrong or missing treatment, etc. Generally, on top of increased suffering and possible ill-health effects on the patient, misdiagnoses cost our society large amounts of money. Unnecessary or wrong treatments, leading to more visits and/or longer stays at the hospital, followed by new x-ray examinations, are costs that will leave marks in the hospital’s budget. Other costs concerning public finance are more difficult to analyze, include missing labour and costs associated with transports to and from the hospital.

Approximately 6.3 million x-ray examinations are made every year in Sweden. Considerable resources are allocated to ensure safe and accurate radiological reporting. Consequently, in order to minimize the risk of misdiagnoses, x-ray images are examined twice, by different doctors, on a routine basis.

Studies indicate [14] that there is a 3–18% deviation rate between initial and final diagnoses, where the higher numbers correspond to advanced diagnoses within e.g. neurology. The Swedish National Board of Health and Welfare (Socialstyrelsen) states that any deviation shall be studied and carefully analyzed to prevent the mistake from being repeated. However prior to eCare’s Feedback product, there has been no support product to aid in this process, and deviations in diagnoses made in radiological units have not been analyzed systematically [23]. This may come as a surprise given that Sweden is perhaps world-leading in putting digital x-ray imaging to use.

Feedback has been awarded correspondingly, including the award for best support product of the year from the prestigious industry journal Dagens Medicin [46].

\(^1\)That is, observing and diagnosing shapes rather than for example a numeric quantity.
1.2 The Feedback System

1.2.1 Overview

On the eCare webpage [80], Feedback is defined using the following:

"Feedback is a web-based system for collaboration, quality assurance and improving quality in work. The system includes a communication system for collaboration among specialists and electronic cooperation between primary care and hospital specialists. The clinical Feedback system also provides the data needed to measure quality in healthcare. Beyond making data from patient records and image analysis systems available to more people who need them and in more contexts, Feedback supports the integration of existing IT systems."

The typical work-flow is this:

1. Doctor \(a\) makes a preliminary diagnosis and feeds it into the Radiology Information System, RIS. The x-ray image is stored in a Picture Archive and Communication System, PACS.

2. Doctor \(b\) registers a secondary (definite) diagnosis in the RIS. If the diagnosis differs from that of doctor \(a\), a discrepancy is registered and graded with a level of severity \(\alpha_0\sim\alpha_3\), where \(\alpha_3\) is the most severe.

3. Feedback queries the RIS and discovers the discrepancy. All relevant data surrounding the case is extracted from the RIS and stored in Feedback’s database.

4. Feedback sends an e-mail to doctor \(a\), asking him or her to log into Feedback to view details about the discrepancy.

5. Similarly, doctor \(b\) is informed, and Feedback is then used as a communication platform, using text or by manipulating the x-ray image (e.g. drawing markers).

This functionality enables doctors to communicate and collaborate also between regions, learn by their mistakes, and consequently increase the quality of x-ray diagnosis-making. Further, since detailed information on all discrepancies are registered in Feedback’s database, Feedback is valuable in research and for generating statistics. As an example, management can use the statistics to find segments in which discrepancies are frequent and measures need to be taken (e.g. staff training).

Four screen-shots illustrating some, but far from all functionality of Feedback is presented in Figures 1.1 to 1.4.

With the next generation of Feedback servers, currently under development, servers will collaborate and exchange statistics on a national basis. This enables for comparison of the individual, a group or doctors or the whole clinic to regional or national standards, which is an extremely valuable resource in the work of quality assurance.

Also, individual discrepancies can be shared nationally, for research or to take advantage of remote expertise.

1.2.2 This Thesis

This thesis addresses two distinct technical functions that are core components Feedback:

- Directory services
- Data propagation
1.2 The Feedback System

How to design and implement these two functions in the context of the Feedback system is the core of this thesis.

The following are my definitions.

**Directory Service**

In order to share data, Feedback servers need to find each other on the computer network. Namely, Feedback servers need ways of translating organizational names (e.g. Stockholms läns landsting / Karolinska Sjukhuset) to network addresses. This is a classic problem that has kept computer engineers busy for as many years as there has been computer networks. Solutions to this problem typically involve some kind of automated directory services, and are consequently referred to as *naming* or *directory services*.

There are many requirements of the directory service to be used within the Feedback application. The service needs to be highly reliable, be able to cope with high load and a large number of entries, and needs to cope with network topology changes, as well as new Feedback servers being added dynamically and existing Feedback servers leaving the network without warning. Ideally, the service will be self-configuring, requiring no human configuration updates in such situations.

Is this possible, given existing technologies, or is a proprietary solution necessary? If so, how will this service be designed and implemented?

**Data Propagation**

Once the Feedback servers have found each other, they need means of sharing data over the network. Once again, this has to be reliable, secure and efficient. How do we do this is not obvious. How do we design and implement this?

Feedback installations also need means of communicating with, or rather extract data from, *ris* and *pacs*. The former (*ris*) type of connection will be discussed in this thesis.

How does Feedback integrate well with legacy *ris* applications? Can we ensure reliability and efficiency?

**The Layout of this Thesis**

As a basis for further reasoning, Chapter 2 will present a broad overview of the current state of distributed computing technology. Next, the problem will be stated in terms of specific requirements in Chapter 3. Chapters 4 to 5 will propose solutions and finally Chapter 6 will conclude and add some final comments.

### 1.2.3 Current Feedback Technical Implementation

Feedback is based on a multi-tier component architecture, mostly implemented in Java using Java RMI as the glue between components. The user utilizes a standard web browser to access the application, e.g. Microsoft Internet Explorer or Netscape Navigator. All communication between the end-user’s browser and Feedback is encrypted for maximum security. The *html* pages are generated dynamically on the Feedback server using a combination of Java Servlets and JavaServer Pages. The business logic tier is based on JavaBeans (but not Enterprise JavaBeans) technology.

At the bottom level, an Oracle database maintains the application data, which consist of discrepancy data, messages, pre-compiled statistical data, and more.
Communication across healthcare units (e.g. hospitals or clinics), which will be implemented in the near future partly based on this thesis, is based on the national IP network sjunet [56], a private TCP/IP network for healthcare units across Sweden.
1.2 The Feedback System

Figure 1.1 The list of discrepancies presented to the doctor at login. Microsoft Internet Explorer v6.0 under Windows 2000.

Figure 1.2 The detailed view of a discrepancy with the preliminary and definite diagnosis and the x-ray image. Microsoft Internet Explorer v6.0 under Windows 2000.
Figure 1.3 The first screen of the statistics-wizard that allows the user to compile and view statistics in any conceivable way. Microsoft Internet Explorer v6.0 under Windows 2000.

Figure 1.4 An example of Feedback generated statistics. Microsoft Internet Explorer v6.0 under Windows 2000.
Chapter 2

Distributed Computing: an Overview

Feedback is an example of a distributed application, that is, an application running in parallel on several physically remote computers. Many technologies already exist to aid in developing such applications, and it is important for the developer to have a broad understanding of existing technology in order not to re-invent the wheel.

Consequently, prior to stating the problem in more detail, as will be done in Chapter 3, and prior to discussing possible solutions in later chapters, distributed computing of today needs to be defined. This will be done throughout this chapter.

2.1 Distributed Computing in a Context

The first distributed extensions to the UNIX operating system emerged in 1970s, however not until late 1980s products and technologies began to emerge that support distributed computing as we know it today [49].

As will soon be discussed, distributed computing introduces a fair number of issues that need to be resolved, problems that are often non-existent in single-computer environments. So why bother? Four main drives for distributed applications are [7]:

Data are distributed
Sometimes, the data an application needs to access resides on multiple computers. Whatever the reasons for this — administrative, operational, legal or historical — the application may be forced to execute on multiple computers to gain access to the data.

Computation is distributed
Some applications execute on multiple computers in order to take advantage of processing power in parallel to solve a problem. Other applications may execute on multiple computers to take advantage of some unique feature of a particular system.

Users are distributed
If users are geographically separate, and need to share data, the application they run must be too. Typically, each user executes a small piece of the distributed application on his or her computer, and shared objects and heavy computations execute on one or more servers.
The advantages outweigh the difficulties and the costs

Below, advantages as well as difficulties with distributed applications are discussed. One or several of these advantages will in many cases simply outweight the costs of addressing the difficulties.

2.2 General Advantages of Distributed Applications

2.2.1 Fault Tolerance

In distributed systems, high levels of fault tolerance can be reached by replicating critical servers or server functions. Ideally, the system will utilize all elements when everything is working normally, then switch to the remaining should one or several crash.

2.2.2 Scalability

A high scalability factor means an application is deployable in a wide range of sizes and configurations. There are two basic strategies to solve the problem of high demand on a shared resource: increase the capacity of the resource, or replicate the resource [49]. Well designed distributed applications are flexible in their deployment and allow replication with better results than in a centralized environment.

2.2.3 Minimized Network Load

At first thought, one might think that increased distribution of processing and data would increase network load. How could network usage possibly be decreased by introducing more server nodes, each one having to be fed with data over the network?

This is best illustrated with an example. Consider an application serving a large number of users with image files. Unless all users take turns in physically sitting in front of the server node, the application must be distributed (since users are distributed). A typical implementation would be a client-server model with thin clients, for example web browsers. A single server node however would impose heavy load on the networking infrastructure. By replicating all or some data at several locations, image data would not have to travel further than between the client and her “nearest” server. New images would have to travel only once between the main server and the local servers. In total, a setup like this has the potential of vastly reducing network load.

2.3 Difficulties with Distributed Applications

2.3.1 Data Integrity

As soon as data is replicated, mechanisms has to be deployed to maintain integrity of the data. In the previous example from section 2.2.3, the local servers “mirroring” the data of the main server need to know whether the data they deliver to clients is still valid or has been replaced by newer data at the main server.

In a more advanced setup, clients will not only read but also add and change data, simultaneously and probably without being aware of each other. In centralized applications, developers have since long used various locking mechanisms to ensure consistency in data. Locking of shared resources is most often handled by the underlying data management systems and is then referred to as atomic transactions [49].
2.4 Supporting Distributed Applications — Environments and Protocols

When distributing the application, the problem gets more difficult to handle. The distributed equivalent of a centralized environment’s atomic transactions is performed by a distributed transaction monitor [49] and plays an important role in many distributed applications. Most modern distributed environments, for example corba [31], Enterprise Java Beans [25], Microsoft .net [36, 27] and most Message-Oriented Middleware implementations [73] support distributed transactions. Further details on these technologies are presented in Section 2.4.4.

2.3.2 Security

Distributed applications rely on a infrastructure, some sort of software on top of a a computer network, to carry messages between components. Such a setup is more vulnerable to eavesdropping and attempts to manipulate, add or destroy data than an application running on a single server node where physical access may be the only way to get close to the data.

Developers of distributed applications need to ensure that [49]:

- All data being sent on a vulnerable channel (e.g. streamed over Internet) is encrypted to prevent eavesdropping and manipulation of data.
- A strong authentication scheme is established to ensure that the identity of users as well as system components can be reliably ascertained.
- The facility exist to log every operation if desired.

2.3.3 Complexity

A distributed system is often a collection of heterogeneous components, logically and possibly physically separate. The components must communicate with each other in a reliable and efficient way, a task that can be trivial or exceedingly difficult depending on what infrastructure the application is built upon.

Also, shared resources must be named without room for ambiguity or misinterpretation. Again, this task often handled by some form of infrastructure, of which we will see examples in the next section.

2.4 Supporting Distributed Applications — Environments and Protocols

A very important feature of distributed computing is the infrastructure that enables the geographically distributed components compromising the system to communicate and collaborate. Several classes of infrastructure, often referred to as ”distributed middleware”, in this thesis referred to as “middleware”, have emerged over the last 15–20 years. The classification used in this thesis, sorted chronologically as they emerged, is:

- sql-based products
- Remote Procedure Calls and Remote Method Invocation
- Distributed Transaction Processing Monitors
- Object Request Brokers and corba
- Message-Oriented Middleware

A brief introduction to these technologies will be presented shortly, but first some terms need to be defined:
Synchronous communication  In synchronous communication, once a module makes a request, execution is blocked for the duration of the call. Execution continues as soon as a reply is received from the module servicing the request.

Asynchronous communication  In asynchronous communication, the calling module carries out other work after the request is sent. The reply is then pushed back upon the calling module, which will choose how and when to handle it. Asynchronous communication often involves multi-threaded programming and can be perceived as difficult to programmers not used to concurrency. However, asynchronous communication gives the ability to overlap computation and communication which can have significantly positive effects on system performance.

Coupling  The concept of coupling refers to the strength of interconnection between two software components; the higher the strength of interconnection, the higher the coupling. For software to be easy to understand and maintain, coupling should be kept as loose as possible [49]. If modules are highly coupled, there is high probability that a programmer modifying one module will need to make subsequent changes in another.

The concept of coupling was introduced by Yourdon and Constantine in [52]. Their original work was presented in 1979, when distributed applications as we know them today did not exist. Rather, Yourdon and Constantine referred to partitioning modules of large but single-computer applications. Today, when applications often consist of many heterogenous components distributed over several physical computers, partitioning the application for minimum coupling is a top priority and fundamental for the long-term success of every distributed application.

Software modules communicating in a synchronous manner are considered to introduce higher levels of coupling than their asynchronous counterparts [49].

2.4.1 sq1-based Products

In early distributed environments, communication between client components (often run on UNIX workstations or terminals) and a central database was achieved by running sq1 commands over the network. Gateway products have later added an abstraction layer on top of database implementations to consolidate different physical databases into a single logical one. Also, the development of procedural capabilities in sq1 resulted in the availability of “stored procedures” in the database, reducing coupling by isolating the clients from the physical layout of the stored data.

2.4.2 Remote Procedure Calls and Remote Method Invocation

The Remote Procedure Call (rpc), is a procedure call executed on a physically remote server. There are similarities to sq1 stored procedures, but rpc is a more general solution, not restricted to querying a relational database. The calling modules makes what appears to be a local call — however the middleware, using its knowledge of the location of the service provider module, takes the arguments of the call, routes them to the destination, and routes the response back to the origin.

The procedures are usually defined in an Interface Definition Language (idl), which differs in syntax between rpc product families. In general terms, an idl is a language that lets a program or object written in one language communicate with another program written in an unknown language.
The main actor on the \texttt{rpc} field is the Distributed Computing Environment (\texttt{dce}) [60] from the Open Group (former Open Software Foundation) [61]. The \texttt{dce}, offering several other features in addition to \texttt{rpc}, is currently ported to most major platforms and has been on the arena since 1992.

Another currently emerging standard is \texttt{xml-rpc} [78, 79], a specification and set of implementations using \texttt{http} as the transport and \texttt{xml} as the encoding.

Most \texttt{rpc} implementations (including \texttt{dce} and \texttt{xml-rpc}) are based on synchronous communication.

A “problem” with \texttt{rpc} is that it does not translate well into object-oriented distributed systems, where communication between program-level objects rather than procedures are needed. To match the semantics of object invocation, Remote Method Invocation (\texttt{rmi}) can be used instead. The Java platform [69] supports \texttt{rmi} through its Java \texttt{rmi} system [70], which is the base platform of the current Feedback implementation. Java \texttt{rmi} is responsible for:

- Locating remote objects.
- Communicating with remote objects (i.e. running remote methods).
- Loading class bytecodes for objects that are passed as parameters or return values.

### 2.4.3 Distributed Transaction Processing Monitors

Some systems operate on top of \texttt{rpc} environments like \texttt{dce}, providing additional capabilities such as distributed transaction monitors. Example implementations are \texttt{cics} [57] from \texttt{ibm} and Tuxedo [55] from \texttt{bea Systems}.

### 2.4.4 Object Request Brokers and \texttt{corba}

An Object Request Broker (\texttt{ora}) provides, similarly to \texttt{rmi}, an infrastructure for distributed objects to communicate. However, standards like \texttt{corba} push things a bit further. Using such technologies, \texttt{ora} implementations from different vendors can collaborate and discover each other’s services. An \texttt{ora} from any vendor, on almost any computer, operating system and network, can interoperate with an \texttt{ora} from the same or another vendor, on almost any other computer, operating system and network. Furthermore, it does not matter what language the program-level objects are written in as long as the \texttt{ora} knows how to handle them.

For an illustrative example, refer to Figure 2.1. The \texttt{ora} is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. The interface presented to the client is completely independent of where the object is located, what programming language it is implemented in, or any other aspect that is not reflected in the object’s interface.

![Figure 2.1 A request being sent through the \texttt{ora}.](image-url)
For each object type, you define an interface in an Interface Definition Language (IDL). Similarly to RPC, the ORB uses the interface to "broker" communication between one object and another. Using abstract interfaces to isolate actual object implementations from one another is a feature aiming at loose coupling (refer to Section 2.4). Similarly to when developing non-distributed object-oriented systems, this allows changes in the implementation without need for subsequent changes in other components as long as the interface remains the same.

CORBA [31], short for Common Object Request Broker Architecture, is the Object Management Group's (OMG) [59] framework for inter-ORB communication. CORBA is today's major standard in this area [49] and most ORB vendors support the standard.

The reference model of the CORBA architecture consists of the following components [31]:

- ORB Architecture and Specifications.
- Object Services, a collection of services that support basic functions for using and implementing objects, e.g., the Life Cycle Service that defines conventions for creating, deleting, copying and moving objects.
- Common Facilities, a collection of services that many applications may share but which are not as fundamental as Object Services, e.g., an electronic mail facility.
- Application Objects, corresponding to the traditional notion of applications, thus not standardized by OMG. Application Objects constitute the uppermost layer of the Reference Model.

**ORB Architecture and Specifications**

A very important part of the CORBA ORB specification is the Interface Definition Language for CORBA, OMG IDL [63]. Interfaces for CORBA's standard Object Services and Common Facilities are all specified in OMG IDL. As illustrated in Section 2.4.4, Application Objects also communicate via interfaces defined in IDL.

A minimalistic example OMG IDL definition [42] may look like this:

```idl
interface sales_tax {
    float calculate_tax ( in float taxable_amount );
}
```

This is an interface to an object of type sales_tax that performs one operation: calculate_tax. The operation takes one input parameter, of type float, and returns a float with the calculated results. Clients interested in communicating with a sales_tax object use the above OMG IDL definition to determine what the ORB expects in terms of name and parameters in order to handle the request.

However names and parameters can also be discovered dynamically using the ORB's dynamic invocation interface. This is a feature that reduces coupling. Why?

Using static interfaces is a good start. However if ever changing the interface, the IDL has to be redefined and clients need to be updated in accordance to the new IDL. When operation names and their parameters are discovered dynamically instead, not only the implementation but also the interface can be changed with no or minor ripple effects through the rest of the system.

For low-level inter-ORB communication, CORBA uses a network protocol called Internet Inter-ORB Protocol (IIOP). The protocol runs over several transport layers, including TCP/IP and rpx/spx. Objects are addressed using Interoperable Object References (IOR), where parts of the address is based on the server's network identity (e.g., IP address and port number) and the rest is used by the ORB to locate the specific object within the set of local objects available.
Object Services

To date, the CORBA standard (version 2.4) specifies the Object Services listed in Table 2.1.

<table>
<thead>
<tr>
<th>Collection Service</th>
<th>Persistent Object Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency Service</td>
<td>Property Service</td>
</tr>
<tr>
<td>Event Service</td>
<td>Query Service</td>
</tr>
<tr>
<td>Externalization Service</td>
<td>Relationship Service</td>
</tr>
<tr>
<td>Naming Service</td>
<td>Security Service</td>
</tr>
<tr>
<td>Licensing Service</td>
<td>Time Service</td>
</tr>
<tr>
<td>Life Cycle Service</td>
<td>Trading Object Service</td>
</tr>
<tr>
<td>Notification Service</td>
<td>Transaction Service</td>
</tr>
</tbody>
</table>

Table 2.1 CORBA 2.4 Object Services

An ORB however does not have to provide all services, and few commercial ORB implementations do. Detailed descriptions of all CORBA Object Services are beyond the scope of this paper, however some interesting observations can be done just by looking at the list. To be begin with, Event Service and Notification Service implies that CORBA handles asynchronous communication, which is true. The Naming Service is an important component that will be discussed in more detail in Section 2.4.8. Also note that CORBA defines a service for distributed transactions, the Transaction Service.

According to OMG themselves, CORBA ORBs and firewalls currently have a limited form of “peaceful co-existence” that provides satisfactory functionality only in some cases [30]. The essential problem with CORBA’s IIOP protocol and firewalls is that it is not easy to know in advance (and to represent in a firewall configuration) which hosts and ports will be used for CORBA communication. The host and port addressing information is contained in ior references that describe how to communicate with servers. It has traditionally been assumed that clients can contact servers directly, at any port, which is not the case when there are network firewalls deployed between the client and the server.

OMG has suggested a number of solutions to this problem, including using a limited number of pre-defined ports and using the socks [22] protocol, which is a protocol for negotiating security policies with a firewall. Originally adopted in late 1998, the firewall specification [30, 32] is still undergoing a major revision when this is written in December, 2001.

Most CORBA ORB vendors will implement their own set of solutions to overcome the difficulties with IIOP and firewalls. Typically, the range of ports used for IIOP traffic can be narrowed into a few or even a single port. Also, some vendors have developed their own IIOP proxy servers for use with existing firewalls.

Java Enterprise Edition and Enterprise JavaBeans

Originated by Sun Microsystems, J2EE (Java 2 Platform, Enterprise Edition) is the Java platform designed for large-scale, distributed computing typical of enterprises. The platform aims at simplifying application development by enabling the tier to handle many aspects of distributed programming automatically, e.g. distributed transactions, concurrency management, and by providing an extensive security model. In this sense there are many similarities to CORBA. However there are also fundamental differences between CORBA and J2EE: J2EE is tightly tied to Java, whereas CORBA is language neutral. It is possible to have J2EE use CORBA to link to other languages, but that is not the same as having integrated support for multiple languages. Besides, not all J2EE implementations support CORBA connectivity.
J2EE applications are made up of components. A J2EE component is a software unit that is assembled in a J2EE application and communicates with other components using the underlying J2EE infrastructure. The J2EE specification defines the following component layers [58]:

**Client components** Applications and applets.

**Web components** Java Servlets and JavaServer Pages (JSP).

**Business components** Enterprise JavaBeans (EJB).

This layered view of the J2EE platform shows another important difference between J2EE and CORBA. Specifications of implementing languages, such as JavaServer Pages above, is beyond the scope of the CORBA standard. CORBA is an *integration* technology, not a *programming* technology — J2EE aims at being both. J2EE components are written in the Java language and compiled in the same way as any other program. The difference is that J2EE components are assembled into a J2EE application, verified that they are well-formed and in compliance with the J2EE specification, then deployed to production where they are run and managed by the J2EE server.

J2EE is a standard, not a product. You cannot “download” J2EE, rather you download a set of documents which describe agreements between components and the J2EE containers in which they run. So long as both sides obey the J2EE contracts, applications can be deployed in a variety of J2EE server implementations. Such implementations are available from a large number of vendors, including Sybase, Fujitsu, IBM and Oracle Corporation. Sun Microsystems also ships a reference implementation of J2EE, however suited for compliance testing rather than deployment [58].

Java Servlets and JavaServer Pages are meant to create HTML, or other formatted data, for the client. Further details about these technologies are beyond the scope of this paper. Please recall from Section 1.2.3 however that parts of the Feedback server is implemented using Java Servlets and JavaServer Pages technologies.

Enterprise JavaBeans, EJB, is the core of J2EE. The EJB specification [25] defines two types of enterprise bean objects:

- A session object.
- An entity object.

A typical session object executes on behalf of a single client, making computations and accessing and updating data on behalf of the client. An entity object typically provides an object view of some data in the database, allowing shared access from multiple users. The EJB specification provides a framework of Java classes the developer extends, overriding skeleton Java methods to implement the desired behavior.

Similarly to CORBA, J2EE offers object services, perhaps most importantly the already mentioned transaction services, concurrency services and security services. Synchronous and asynchronous communication is supported, and through the Java Message Service, asynchronous message-driven communication (more on this topic in Section 2.4.5). However the services delivered by J2EE are not as complete and mature as the CORBA object services, and there is a much smaller set of them.

Interestingly enough, Sun Microsystems has adopted the CORBA Transaction Service as the basis for the Java Transaction Service (JTS). Meanwhile, OMG has moved rapidly to incorporate further support for Java into its architecture. There has, for example, been interest in the OMG community in adopting a component model consistent with the Enterprise JavaBeans component model. Possibly and hopefully CORBA and J2EE will eventually merge to produce a combined capability whose whole is more powerful than the sum of its parts.
2.4 Supporting Distributed Applications — Environments and Protocols

**Microsoft .NET**

Microsoft .NET from Microsoft Corporation, still at an early stage in its life-cycle, is a product suite with many similarities to J2EE — the distributed application runs in a server container that provides services enabling the developer to focus on business rules rather than the nuts and bolts of distributed computing.

Microsoft .NET is largely a rewrite of Microsoft Windows Distributed interNet Applications Architecture (Windows DNA), Microsoft’s previous platform for developing enterprise applications. Windows DNA includes many technologies that are in production today, including Microsoft Transaction Server (MTS) and COM+, Microsoft Message Queue (MSMQ), and the Microsoft SQL Server database. The new .NET platform replaces these technologies, and more.

Following is an itemized list of the technical components making up the Microsoft .NET platform [12]:

- **C#** A new language for writing components, integrating elements of C, C++ and Java, and adding some additional features.

- **IL** A “common language runtime”, which runs bytecodes in an Internal Language (IL) format, similarly to Java bytecode and the Java Virtual Machine.

**Base components** Providing various functions similarly to CORBA Object Services.

- **ASP+** A new version of ASP that supports compilation of ASP pages into IL, similarly to JavaServer Pages.

**Win Forms and Web Forms** New user interface components accessible from Microsoft’s development tools.

- **ADO+** A new generation of ActiveX Data Objects (ADO) components, built on the premise of XML-based data interchange, facilitating access to relational or non-relational databases.

The Microsoft .NET core runs on Microsoft Windows only but in theory supports development in many languages — as soon as IL compilers have been created for them. Components communicate over Simple Object Access Protocol (SOAP) [6], an open HTTP and XML based initiative put forward by the World Wide Web consortium [77]. Although SOAP is a new and virtually un-tested standard, the initiative to base the product on SOAP means .NET is open to non-Microsoft .NET components.

Comparing Microsoft .NET and J2EE, it is important to note that while J2EE is a standard, Microsoft .NET is a product range. Using J2EE technology, a large number of J2EE server implementations, on a large number of platforms, is available. The danger however in an open standard like J2EE is that if vendors are not held strictly to the standard, application portability is sacrificed. To help with the situation, Sun Microsystems has built a J2EE compatibility test suite [67], ensuring J2EE platforms comply with the standards.

Microsoft .NET on the other hand provides a solution, complete or not, from a single vendor — Microsoft. This way there is no product portability at all. How much of the .NET framework will be available on other platforms? Even with an open protocol like SOAP, deployment of applications containing Microsoft .NET components will never be flexible as long as the core .NET services run only on the Microsoft Windows family of operating systems.

### 2.4.5 Message-Oriented Middleware

The final class of middleware presented in this thesis, the Message-Oriented Middleware ( MOM), is based on distributed communication that already by definition is loosely coupled.
Using asynchronous, message-based communication channels that guarantees message delivery, applications are completely isolated from communication networks which makes applications simpler and shielded from network changes. More specifically, a messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending and receiving messages.

The sender and receiver do not have to be available at the same time in order to communicate, neither do the sender or receiver need to know anything about each other. Using a publish/subscribe mom product, publishers and subscribers may dynamically publish or subscribe to a topic, and the mom takes care of distributing the messages around the network.

Most mom products also allow point-to-point messaging, where every message has only one consumer. Messages are still addressed to a queue administered by the mom, so the non-available and asynchronous characteristics remain. Queues retain all messages sent to them until the messages are consumed or until they expire.

Many or most mom vendors have also implemented distributed transaction monitors in their products.

Example Products

**mqseries** mqueue series from ibm enable mom-services through its Message Queue Interface, available for many programming languages including Java and c++. The product supports distributed transactions and is available on over 35 platforms [65]. mqueue clients, with read-only capabilities, are free of charge. mqueue servers, with read and write capabilities, are priced in accordance to the processing power of the server they are to be deployed at. A server node with 1–2 risc processors is valued 2 capacity units [65], each capacity unit being priced at 14.500 sek through the Swedish Stadskontorsavtalet, >22.000 sek otherwise (1 October 2001).

**TIB/Rendezvous** TIBCO’s TIB/Rendezvous [76] product is the messaging system that is the foundation of TIBCO’s line of e-business infrastructure products. The product supports distributed transactions and can be used for non-available, asynchronous messaging as well as synchronous request/reply communication. Messages are carried between networks by TIB/Rendezvous’ software routers”.

**Oracle Advanced Queueing** Oracle Advanced Queueing is a database-integrated message queueing system, integrating asynchronous messaging into the database itself. The queueing service can be accessed using c/c++, Visual Basic, Oracle’s pl/sql or Java via jms [68] (see below). Advanced Queueing also fits neatly together with several other mom products, including TIB/Rendezvous.

Queues can be distributed across servers and networks by employing Advanced Queueing “hubs” routing messages between Advanced Queueing servers [41].

**Java Messaging Service** The Java Message Service (JMS) is a Java api that allows Java applications to communicate with mom middleware. Supported products are, among many others, all mom products mentioned in this paper. The designers at Sun Microsystems have chosen a lowest common denominator approach, striving to maximize the portability of JMS applications across mom providers. This comes with a side-effect however; some vendors, including Oracle Corporation, have developed their own supersets of JMS, taking full advantage of their respective mom products. Yet portability is not in danger as long as a standard JMS is available for those who are willing to sacrifice some extended functionality for guaranteed portability.
2.4 Supporting Distributed Applications — Environments and Protocols

JMS is integrated into J2EE from J2EE version 1.3 and onwards. It can also be downloaded as a separate extension.

2.4.6 Encryption and Authentication

Ciphers and x.509

Cryptology, the art of devising and breaking ciphers, has a long history. One of the oldest known cryptographical methods is known as the Caesar cipher, attributed to Julius Caesar, used by the Romans some 2000 years ago. The Caesar cipher is a symmetric key cipher, which means that the message to be encrypted (the plaintext) is transformed by a function parameterised by a secret key, producing an encrypted ciphertext. The encryption function in the Caesar cipher was shifting letters — the secret key was the number of shifts to perform. Ciphertexts can be transformed back to plaintext by applying a reverse version of the function using the same secret key.

Modern symmetric key cipher cryptography uses largely the same ideas as traditional cryptography, although the encryption functions are extremely complex and the keys are longer. The key length is important, since larger keys increase the work factor needed to break the cipher by exhaustive search (trial-and-error). Some of the more commonly used symmetric ciphers of today are DES [2], 3DES and IDEA [54] with key lengths of 56 bits, 168 bits and 128 bits respectively. The latter two, 3DES and IDEA, are generally considered safe, whereas DES is no longer adequate [43]. In reality however DES is still used for secure applications such as for example banking using automated teller machines [43].

In 1976, researchers Diffie and Hellman proposed a radically new kind of cryptosystem [11], where the encryption and decryption keys were different, and the decryption key could not be derived from the encryption key. The encryption key could then be made public, whereas the decryption key would be kept secret — hence the name public-key cryptography or asymmetric key ciphers.

Two years later, a research group at M.I.T discovered an asymmetric key cipher method later named by the initials of the three discoverers: RSA [40]. This method has an important feature. To begin with, the method has the property suggested by Diffie and Hellman, that is:

\[ D(E(P)) = P \]  \hspace{1cm} (2.1)

... meaning that applying the public-key, then the secret-key on the plaintext yields the plaintext in its original form. The RSA algorithm however also has the following characteristics:

\[ E(D(P)) = P \]  \hspace{1cm} (2.2)

That is, applying the secret-key, then the public-key, yields the original text just as applying the keys in the reversed order as shown above. This feature is commonly used for electronic signing of documents. How?

Take an example of Alice sending a message \( P \) to Bob. She starts by encrypting the message with her secret-key. She then encrypts the resulting ciphertext with Bob’s public-key before sending it to Bob. Later, Bob receives the message and decrypts it with his private-key. This yields the plaintext \( P \) encrypted with Alice’s secret-key. Bob already has a copy of Alice’s public-key and can extract the original plaintext by finally applying Alice’s public-key.

1In real applications, Alice will not often sign the entire document but rather a message-digest of the document, that is, a short fingerprint generated using a well-known algorithm. She encrypts the whole document only once, using Bob’s public key, and attaches a signed copy of the document’s fingerprint. To validate the signature, Bob extracts Alice’s signature using her public-key, then calculates the fingerprint himself and compares it to the fingerprint Alice
Bob is now certain that Alice sent him this message since she is the only one in control of her secret-key, and correspondingly she is the only one who can produce ciphers that are decryptable with her public-key!

A widely used application of these signing models are the x.509 certificates — digital identities, commonly used for authentication of hosts, applications or application components. Certificates are issued by a trusted certification authority (CA) and contain the owner’s public-key, her identity, and an expiration date. The CA signs the certificate using its private-key. Then, using the public-key of the CA, a user can verify that the document is signed with the CA’s private-key. This, of course, assumes that the receiver has the public-key of the CA. Correspondingly, public-keys from several commercial CA’s are bundled with Netscape Navigator as well as Microsoft Internet Explorer and several e-mail clients.

**TLS/SSL and IPsec**

SSL [16] (Secure Sockets Layer), its successor TLS [10] (Transport Layer Security) and IPsec [21] (IP Security Protocol) are today’s most widely used mechanisms for encryption of data and authentication of users and system components. They do however attack the problems in different ways.

IPsec is a technology enabling packet encryption and authentication at the network (IP) layer of the network and originates from the upcoming IPv6 standard. IPsec is generally used between IP routers to establish virtual private networks, site-to-site tunnelled connections, e.g. securely binding two branch offices (Stockholm and Gothenburg) over an insecure connection (Internet). The transport layer protocols on client machines (e.g. TCP or UDP) are unaware of the underlying encryption and/or authentication schemes. IPsec can also be used for machine-to-machine or machine-to-site tunnelling. Then, the requirement is that every endpoint, whether a server node or a dedicated hardware or software router, is configured an IPsec router.

IPsec can be used with many authentication and encryption algorithms, including authentication based on x.509 certificates. Several modern operating systems already include x.509 support and IPsec implementations. Additionally, third party implementations are available for most server-oriented operating systems in use today.

It is important to note that IPsec only protect from outsiders, that is, hosts outside the IPsec router. Within the network, traffic is not secured.

SSL and TLS both work in the application layer, which means applications must be tailored to use SSL or TLS. The protocols are integral parts of most web browsers and web servers. They are, however, not restricted for use in combination with the HTTP protocol.

SSL has recently been succeeded by TLS, which is based on SSL and sometimes referred to as SSL version 3.1. Although TLS is not backward-compatible, TLS enabled applications always tend to support the older SSL as well.

Strong authentication mechanisms is supported in TLS and SSL through the use of x.509 certificates, and the specifications include a mandatory set of encryption ciphers. Implementations however are welcome to include more than the minimum set of ciphers, and most do.

Many middleware implementations, perhaps most notably from the MOM and ORB classes, will let all network traffic be streamed over TLS or SSL if desired.

---

sent him.

Signing message digests rather than entire documents is for performance reasons as RSA encryption/decryption is very expensive.
2.4 Supporting Distributed Applications — Environments and Protocols

A Brief Comparison: TLS or IPSec?

IPSec requires configuration changes at network level. Also, IPSec tunnels have problems with certain types of firewalls\(^2\) between the IPSec routers.

**TLS** and **SSL** can be integrated in applications even where the developer has no control over the network and **TLS/SSL** survive IP masquerading firewalls. The protocols also protect from both insiders as well as outsiders. However, applications need to be modified to use **TLS/SSL**, which limits the use of these standards with legacy applications.

Since **IPsec** is completely transparent at application level, **IPsec** is a good choice when applications have not or can not be upgraded to use **TLS/SSL**.

2.4.7 Choosing Middleware and Appropriate Level of Coupling

To this point, this chapter has presented an overview and a rough classification of today’s middleware technologies. The term “coupling” was defined in Section 2.4, and the middleware technologies presented have been ordered from tight coupling (synchronous SQL and Remote Procedure Calls) to loose coupling (asynchronous messaging). Interestingly enough, the ordering is also chronological.

So, since development of middleware has been progressing towards lower levels of coupling, can and should loose coupling always be strived for when designing an application? The short answer is yes. However, some degree of coupling is always necessarily introduced in a distributed system, and there is always a minimum level of coupling needed to avoid data integrity problems [49]. Why?

Take, for example, an online shopping site where a software module for the “process order” task requires the customer’s credit status to be checked, each items on the order to be accepted and the total discount to be calculated, all from different remote functions. Until one function in this sequence performs its work, the software cannot progress to the next task. Therefore, the software module has so called processing dependencies with each of these remote modules. Implementing such a scenario using asynchronous communication models make little sense, and can even harm the system. Thus, the coupling induced in this application relates to business rules rather than design and implementation issues.

In **SQL** and **RPC**, things are easy since synchronous request/reply communication is all there is. However at the other end of the spectrum, few **MOO** products are designed to be used in such a manner. Designing distributed applications, there are often at least one or two situations where processing dependencies are necessary, and likewise **MOO** middleware is not often used within but rather **between** distributed applications.

**CORBA** and **J2EE** are two very strong candidates when it comes to building the distributed application. Vendor implementations of these standards however most often cost money, sometimes considerable amounts. For those who are happy with nothing else than the Java language, and can afford to live without distributed transactions and asynchronous communication, **Java RMI** is a popular alternative. It is a standard part of the Java platform and completely free of charge.

**Microsoft .NET** is an interesting, emerging alternative but of limited interest to eCare since it only deploys on the Microsoft Windows family of operating systems.

---

\(^2\)Namely, firewalls that perform Network Address Translation (NAT), that is, masquerading a set of internal, private network addresses by always replacing the sender address with its own address when a datagram leaves the private network. Replies from the outside will arrive at the firewall, which will again alter the datagrams by replacing the destination address with a private address and forward the datagram to the internal network for delivery.
2.4.8 Naming and Directory Services

In a computer network, a naming service tells you where in the network something is located. That is, names are associated with objects and vice versa.

Naming services play an important role in all classes of middleware presented in this thesis. Two distinct functions of a naming services can be observed:

- To map human-friendly names to identifiers used internally in applications or computer networks, e.g. DNS, see below.
- To locate and access an object whose identifier is different between times but the name is persistent, e.g. CORBA Naming Service, see below.

A directory service is an extension to a naming service where a name is not only associated with an object but also with a set of attributes describing the object. Apart from enabling clients to look up objects by their name, the directory service provides operations for creating, adding removing and modifying the attributes associated with objects.

The information is generally read more often than it is written, and as a consequence, naming services generally do not implement complicated distributed transactions. Typically, naming services are distributed and server nodes have the ability to replicate information in order to increase availability and decrease response time. Temporary inconsistencies between the replicas may be accepted as long as they get in sync eventually.

Today's directory service standards generally lacks one or several of the following areas, making them difficult to incorporate into the Feedback product:

- Scalability
- Dynamic addition of entries
- Dynamic update of entry data
- Tolerance to network topology changes

Below are presentation of some specific technologies. More specific comments on how they would fit the needs of the Feedback system will be presented in Section 4.2.

Domain Name System (DNS)

The DNS, arguably one of the biggest and most successful directory services today, is a distributed directory service used to map human-friendly domain names (e.g. www.ecare.se) to network-friendly IP addresses (e.g. 195.42.192.187) and vice versa. The service has a uniform namespace, that is, you have the same view of the data no matter where you are in relation to it. The crucial DNS documentation is provided in [28, 29].

The domain name space is a tree structure. There is no distinction between nodes and leaves, and they are both commonly referred to as "nodes". Each node has a label of 0–63 characters, the zero length label however being reserved for the root node.

When a user reads or writes a domain name, the node labels are separated by dots ("."). A domain name is a sum of labels and can be absolute, i.e. starting from the root domain, or relative, omitting all nodes above a given level. Relative domain names must be completed by local software using knowledge of the local domain before being used on the DNS.

The DNS is a fully distributed service that allows replicating as well as partitioning of the global DNS tree over multiple servers. The tree is partitioned into zones, and each zone has
at least one node, and hence a domain name, for which it is authoritative. Zones are logical concepts that can be hosted on one or several physical servers.

When a look-up request is issued, a DNS server traverses the DNS tree starting from the root domain until it finds a server being authoritative for the zone containing the requested domain. Thus, authority of zones has to be delegated from respective parent zones. There are many authorities all over the world being responsible for delegating sub-zones to their respective zones (e.g. com, se and co.uk). However a DNS tree may just as well be used strictly within an organization, with the organization’s own root servers separate from the global DNS.

Somewhat confusing, the protocol used in DNS communication lacks a special name. Thus, the term DNS can refer to the system itself or the protocol that drives it. Further details on the protocol is beyond the scope of this paper.

**Lightweight Directory Access Protocol (LDAP)**

The LDAP is, as opposed to DNS, a general-purpose directory service, proposed as Internet standard by The Internet Engineering Task Force [66, 51, 48]. The standard originates from the DAPE Directory Access Protocol [62], which uses the overly complex OSI stack [5] rather than TCP/IP [38, 37]. LDAP came about as a result for lightweight TCP/IP clients to access DAPE directories. LDAP defines a protocol for accessing and updating directory information, a model defining the form of the information, and a namespace defining how information is referenced and organized.

LDAP assumes there are one or more servers which jointly provide access to a Directory Information Tree (DIT). Every entry in the hierarchical tree carries a Distinguished Name (DN), the equivalent of a filesystem’s absolute path, which is unique in the tree. Each entry must carry at least the attribute objectClass which defines the type of entry, e.g. person, server or organisation. The LDAP standard includes a large number of pre-defined object classes but is extensible with custom object classes.

Please refer to Figure 2.2 for an example of a LDAP directory information tree. The DN for the person object at the bottom is (cn=Svante Hedin, ou=Stockholm, o=eCare, co=Sweden).

```
cn=Sweden
  objectClass=country

  o=eCare
  objectClass=organization

  ou=Stockholm
  objectClass=organizationalUnit

  cn=Svante Hedin
  mail=svante@ecare.se
  objectClass=person

  ou=Göteborg
  objectClass=organizationalUnit
```

**Figure 2.2 An example LDAP tree.**

Flexible and robust models for authentication, authorization and encrypted channels were introduced in LDAP version 3 [48, 47, 19]. Another interesting feature introduced in version 3 is smart referrals, allowing a LDAP server to map a directory entry or a directory tree to a specific LDAP URL. This way, LDAP requests can be mapped to:

- The same or different name spaces on a different server.
Different name spaces on the local server.

Smart referrals are typically used for scaling, load balancing and to keep deployment changes transparent to the users.

The LDAP Data Interchange format (LDIF) [17] is commonly used to import and export directory information between LDAP-based directory servers, or to describe a set of changes which are to be applied to a directory.

Some commonly referred to LDAP server implementations are IBM SecureWay Directory from IBM, Oracle Internet Directory from Oracle Corporation, iPlanet Directory Server from iPlanet and OpenLDAP, an open-source initiative by the LDAP Community. All these implementations are based on LDAP version 3.

SMB and NetBIOS

Server Message Block, or SMB, is a protocol for sharing files, printers, serial ports and communication abstractions such as named pipes and mail slots between computers [26]. The protocol also defines a naming service for clients to find shared resources on the network.

NetBIOS is a standard from IBM, enabling clients on early PC networks to establish and maintain communications. NetBIOS does not in itself support a standard frame or data format for transmission so applications must use a transport mechanism in addition to NetBIOS. A standard frame format is provided in the NetBIOS Extended User Interface (NETBEUI), however NetBIOS can also be used over TCP/IP.

The SMB and NetBIOS standards have been merged into the Microsoft Windows family of operating system as well as IBM's OS/2. For UNIX-like operating systems, there are several alternatives, including an open-source implementation called Samba [75].

NetBIOS naming is based on servers broadcasting their presence on the network. Clients listen for these broadcasts and build browse lists accordingly, binding NetBIOS names to network addresses. These properties make NetBIOS a self-configuring directory service, suitable for local area networks with a limited number of NetBIOS servers. In a larger perspective however, problems arise. Routers do not normally send broadcasts outside the subnet from which they originate. And if they were, the network would be eventually be cluttered with NetBIOS servers reminding the world about their existence.

Using NetBIOS naming in a routed network requires the deployment of Windows Internet Name Service (WINS) servers to carry name claims and queries between subnets. Such services are integrated in Microsoft’s Windows NT and Windows 2000 Server, as well as Samba.

Differently from LDAP and DNS, NetBIOS does not allow for names to be organized in trees, but rather represents all names on the same level.

SMB, being a higher-level service than NetBIOS, uses either NetBIOS, NETBEUI or IPX/SPX for data transport. When run over NetBIOS or NETBEUI, SMB uses NetBIOS naming. Using IPX/SPX, SMB relies on a proprietary naming scheme very similar to NetBIOS naming. Name claims however will be forwarded by IPX routers up to eight (8) hops.

Directory Services Markup Language (DSML)

The Directory Services Markup Language, or DSML, provides means for representing directory information as a XML document. DSML is intended to be a simple XML definition that will enable directories to publish basic profile information in a form that it can be easily shared via native Internet protocols as well as used by other applications.

The DSML does not specify a transfer protocol and is intended to work as an optional output in combination with standards such as for example LDAP [44].
CORBA Naming Service

The Naming Service [34] is one of the CORBA Object Services defined in Section 2.4.4, responsible for binding names to objects. This service can be used in several ways. One obvious usage is to use within CORBA-driven applications to get references to remote components, but the service can also be used as a more generic directory service. Imagine, for example, objects of type feedback_server encapsulating IP addresses of Feedback servers across Sweden.

To resolve a name is to find the object associated with the name in a given naming context. A naming context is an object that contains a set of name bindings, and because a context is like any other object, it can also be bound to a name in another naming context. Binding contexts in other contexts creates a naming graph, allowing objects to be named in a hierarchical tree-like structure.

The naming graph can span multiple Naming Services, which can themselves reside on different hosts. Given the initial (root) naming context of a remote naming service, a naming context can transparently bind itself to that Naming Service’s naming graph. A naming graph spanning multiple naming services is said to be federated.

Again, in the example of a Naming Service populated with feedback_server objects, the objects could be organized in a graph structure corresponding to the “real” organization, e.g. county councils. Further, the respective objects could be hosted at the corresponding physical Feedback servers. In that way, by the time the server is located in the naming graph, a connection is already established.

2.4.9 Data Propagation

Earlier in this thesis, middleware was defined as the infrastructure that enables the geographically distributed components comprising the system to communicate and collaborate. All types of middleware later discussed (e.g. CORBA, J2EE and RPC) define their own models and protocols for low-level data propagation, that is, passing data from one process to another over the network.

Data Marshaling, Protocols and Complexity

The minimalistic approach to sending and receiving data over an IP network is to open a TCP socket and exchange raw binary data over the stream. However possible, this approach is rarely a good option from the application developer’s point of view. The developer is rarely interested in sending or receiving raw bytes — more often, she wants to pass more abstract data such as integers and strings, which are represented differently in different computer systems, or even more abstract data such as programming objects or a collection of objects with references to one and another. When so, the developer can either define her own protocol or choose to rely on the middleware for marshaling the data, that is, converting the collection of items into a form suitable for transmission over a byte stream. At the other end of the network, data is unmarshaled and passed to the application in a form suitable for that specific platform and operating system.

The needs and complexity of data marshalling differs with different classes of middleware. Perhaps the most advanced marshalling is done in ORB-based products, where interfaces can be self-describing and programming objects can have references to object instances on physically remote computers. And increased complexity comes with increased needs for processing power. Thus, it has long been argued whether for example CORBA and its iOR is a good platform when it comes to raw data transfer. The sceptics argue that CORBA is too complex and introduces unnecessary overhead for bulk data transfers. However there are also several
examples where corba has been successfully deployed in streaming media and realtime applications [24, 35].

File Transfers

There are, of course, specialized situations of data transfer where there is no need for a general and flexible solution such as corba or mom. There are for example a large number of protocols designed exclusively to transport files over a network. Two of the perhaps most commonly referred to examples, FTP [39] and HTTP [13], are supported by a large number of applications and utilities designed for the task of transferring files from one computer to another.

Another example is ssh [4], a set of utilities and a protocol to securely transfer files over a network and to get remote access to a computer.

There is nothing stopping distributed applications to use file transfer protocols for within-application data interchange. These protocols are generally simple and straightforward and tend to focus more on speed and efficiency than anything else. They do not, however, marshal the data in any way and are “dumb” in the sense that they do not understand what they are communicating.

Roy T. Fielding from University of California elaborated on the topic of HTTP versus CORBA in the Yahoo! newsgroup new-httpd (March 4, 1999):

“HTTP/1.0 beats CORBA for any operation involving serious data transfer (>2kb payload), because HTTP doesn’t have to discover the interface before using it, doesn’t have to buffer data into smaller interactions, doesn’t need to wait until the parameters are complete before the stub returns, and doesn’t have to perform data marshaling beyond the header fields. Any working API that is designed for simultaneous streaming data transfer of control data, representation data, representation metadata, resource metadata, and meta-metadata will outperform one that requires multiple interactions to achieve the same results, and that is before the performance tuning.

[...] This doesn’t mean CORBA is a bad thing — it just means it wasn’t designed for distributed hypermedia. There are hundreds of other applications for which CORBA-like protocols are better than large-grain data transfer protocols.”
Chapter 3

Requirements

As mentioned in Chapter 1, the Feedback servers need to communicate several types of information:

- Statistics on discrepancies exchanged between Feedback servers.
- Discrepancy and patient data retrieved from RIS.
- Binary data such as x-ray images retrieved from PACS.

In Chapter 1, the requirements for this functionality were stated in terms of two services — directory services and models for data propagation. The need for these two will be discussed in further detail in the next sections.

3.1 Directory Service

Some general requirements for the directory services were sketched already in Chapter 1. Restating these, in somewhat more technical terms, yields the following list:

**Scalability** To ensure room for growth, the service needs to scale well, ideally without limitations. The service should be flexible and scalable enough to last for many years.

**Reliability** The directory service must be made fault-tolerant, not allowing a single point of failure to affect the whole service. Since the national network of Feedback servers will rely on the same directory service, this is extremely important.

**Dynamic addition, removal and updating of entries** Feedback servers will come and go. The service needs to be flexible enough to add and remove entries without interruptions to the service, as well as being able to make changes to already existent data, e.g. change name or IP address of a Feedback server.

**Tolerance to network topology changes** The service must be self-configuring and tolerant to organizational changes in the physical networks.

**Security** The service must be safe from eavesdropping and unauthorized manipulation of entries.

These are the requirements for the Feedback directory service. Is this possible, given existing technologies, or is a proprietary solution necessary? If so, how will this service be designed and implemented? This question is answered in Chapter 4.
3.2 Data Propagation

Sharing data between Feedback servers, and between Feedback servers and RIS or PACS, the following factors need to be considered:

Reliability The propagation of data needs to be robust and tolerant to network deficiencies.

Efficiency Bulk data transfers should be efficient in terms of network usage as well as processing needs.

Minimum coupling Flexible and loosely coupled integration techniques are necessary to make the solution work long-term.

Firewall tolerance Feedback servers will in general be deployed behind one or several layers of network firewalls, something that needs to be considered in designing transfer models.

Security The data propagation mechanisms need to be protected from eavesdropping and unauthorized manipulation of data.

Recommended implementations to reach these requirements are presented in Chapter 5.
Chapter 4

Proposed Solution: Directory Services

This chapter proposes a design and implementation scheme for the Feedback directory services.

Initially, some high-level concepts are defined. Then, existing directory services as described in Chapter 2 will be revisited and evaluated according to the specific requirements of the Feedback directory services.

Finally, Section 4.3 will conclude by presenting short-term and long-term recommended solutions.

4.1 Introducing the Feedback Tree

Following is a suggested high-level model to propagate the statistics, that will form the basis for the Feedback directory services.

Feedback servers will organize themselves in a hierarchical structure corresponding to real organizational and geographical structures. Some of the communication needs will be point-to-point, that is, one Feedback server will query another server and get a response back. The statistics however needs to be collected from the bottom of the graph and up, being summarized at each level in the tree. Take, for example, a lowest level of “hospital”. All hospital-level Feedback servers will send, on request, their statistics to the server in control of the next (upper) level, the county council. Here, statistics for the council will be compiled before passing it on to the next level, the regional server. Eventually, at the top of the graph, a national server will gather statistics from all regions and compile national statistics.

The Feedback systems needs a directory service to organize itself into a system-global (national) graph, spanning all deployed Feedback servers, with clear agreements on what Feedback server is responsible for compiling statistics at each level, in each branch of the graph. This graph will hence be referred to as the Feedback Tree.

4.2 Comparing Existing Directory Services

So how do the directory services presented in Chapter 2 match the overall requirements of the Feedback system, described in detail in Chapter 3, Problem statement? Are they at all suitable for building the Feedback Tree? In the following sections, existing directory service standards will be revisited and evaluated according to these requirements.
4.2.1 Domain Name System

**Scalability**  The DNS is a distributed service which scales well by allowing zones to be split into as many zones as there are levels in the graph.

**Reliability**  The DNS is a mature service, with the possibility of having an arbitrary number of servers host the same zone. All but one of the servers will then act as a mirroring servers, replicating data from the primary server, receiving notifications when data is changed.

**Dynamic addition, removal and updating of entries**  The DNS was originally designed to support queries of a statically configured database. However using a (recent) DNS server implementation that supports dynamic updates [45], addition and removal of entries are supported, but not addition and removal of zones or servers hosting zones.

**Tolerance to network topology changes**  Although entries can be added, removed or changed, there is no support for configuring servers and zones dynamically and hence no tolerance for servers moving around on the network without humans making manual updates.

**Security**  DNS can be used over SSL sockets.

**General conformance to the needs of Feedback**  Due to its scalable but still statically configured nature, DNS is not suited for deployment in building the Feedback Tree. Further, DNS is not a general directory service but rather designed to explicitly translate domain names to IP addresses and back.

4.2.2 LDAP

**Scalability**  The LDAP is a fully distributed service which, using “smart referrals”, scales well across local and wide area networks.

**Reliability**  LDAP is a robust and mature service with implementations available from a large number of vendors. Replication of servers is supported.

**Dynamic addition, removal and updating of entries**  Using the newly proposed extensions for dynamic updates [50], adding, removal and updating of entries are supported. Specified in 1999, the extensions are now implemented in most LDAP server implementations.

**Tolerance to network topology changes**  Using smart referrals, deployment changes can be made transparent to clients. However implementing the Feedback Tree in LDAP would require dynamic control of smart referrals, which is currently not supported. The LDAP server implementations of today rely on static configuring of smart referrals.

**Security**  Recall from Section 2.4.8 that LDAP version 3 introduces flexible and robust models for authentication, authorization and encrypted channels.

**General conformance to the needs of Feedback**  Similarly to DNS, due to its statically configured deployment nature, LDAP is not suited for building the Feedback Tree unless LDAP servers are modified according to the additional requirements.
4.2 Comparing Existing Directory Services

4.2.3 SMB and NetBIOS

Scalability and reliability

NetBIOS is an old protocol, the core documentation [3] being dated 1987, that was never designed to be used beyond local area networks, neither in terms of scalability nor reliability.

NetBIOS and SMB over IPX/SPX do not scale well beyond local area networks. The protocols rely on network broadcasting (network packets addressed to all nodes on a network rather than a single node) which makes them very “chatty” as networks grow large. As stated in Chapter 2, NetBIOS naming in a routed (i.e. beyond a single subnet) network requires the deployment of Windows Internet Name Service (WINS) servers to carry name claims and queries between subnets.

While WINS servers can be replicated, the service they provide is still centralized. Microsoft’s Windows 2000 Server introduces the option of using dynamic DNS rather than NetBIOS for naming in Microsoft Windows networks, which proves that Microsoft, one of the key contributors to SMB and NetBIOS, are themselves migrating from this directory service — very likely for scaling reasons.

Dynamic addition, removal and updating of entries

This is the key strength of SMB and NetBIOS and one of the fundamentals around which the protocols are developed.

Tolerance to network topology changes

Changes within a local area network, e.g. a host changing IP address, is no match for NetBIOS or SMB. Changes at a higher level however, e.g. subnetworks being shifted around, require manual updating of WINS servers.

Security

SMB supports signing of packets, ensuring authenticity of SMB packets sent and received. The protocol does not support encryption of packets however.

The NetBIOS standard provides no support for signing or encryption.

General conformance to the needs of Feedback

NetBIOS is an open protocol — SMB on the other hand, is not. Some specifications [26] can be publicly downloaded from Microsoft, however not the complete set nor the latest versions. In a world of distributed computing where many or most standards are open and publicly discussed and developed, this is not a comforting fact.

For naming alone, NetBIOS can be used. The flaw of such a setup is that NetBIOS does not allow organizing names in trees, but rather represents all names on the same level. This is not acceptable for the Feedback Tree. Further, both SMB and NetBIOS has serious shortcomings in regards of scalability and reliability.

4.2.4 CORBA Naming Service

Scalability

The CORBA Naming Service can, as described in Section 2.4.8, be distributed over an arbitrary number of servers. The naming contexts, representing levels in the tree, are location independent.

Reliability

Most commercial Naming Service implementations support replication of servers and persistent storage of entries, that is, entries are stored in a database to survive the failover of a Naming Service server. CORBA Naming Service is used in many large-scale, mission-critical systems worldwide and must be considered a mature and robust standard.
Dynamic addition, removal and updating of entries  This is one of the fundamentals of the Naming Service.

Tolerance to network topology changes  The Naming Service does not by itself show any tolerance to structural changes in the network. When network addresses change, object references will turn invalid. Tolerance to structural changes in the network has to be implemented by the developer that is using the corba Naming Service to build his or her application.

Security  As already stated, corba’s hop can be tunnelled through ssl for authentication and encryption.

General conformance to the needs of Feedback  The corba Naming Service is not, like ldap or dns, a service where off-the-shelf servers and their corresponding clients are available. corba Naming Service is a tool for developers of distributed applications, and without any obvious flaws that would make it difficult or inappropriate for use in the Feedback system, except for the possible deployment problems with networking firewalls that has been discussed in previous chapters.

4.3 Proposed Solution

Unfortunately, dns, ldap and smb/netbios were all ruled out as suitable candidate platforms for the Feedback Tree. What remains is a proprietary solution, based on corba Naming Service or a proprietary protocol. This service is named Feedback Directory Service, fds, is introduced in this thesis and will be defined throughout the following sections. First, however, the Feedback Tree will be properly defined as a basis for the definition of fds.

4.3.1 Defining the Feedback Tree

The Feedback Tree is defined as a system-global graph, spanning all Feedback servers across Sweden. As was stated in Chapter 3, the graph is used for Feedback servers to find each other on the network, and to aid in propagation of statistics. Since neither dns, ldap or smb/netbios are suitable candidates for building this tree, a custom definition is proposed in this chapter.

Chapter 3 used the concept of “levels”, representing for example hospital level or council level. This definition is now extended to domains, similarly to how dns is structured.

For an example of the structure and of a (small) Feedback Tree, please refer to Figure 4.1.

Note from the figure that Feedback Tree semantics distinguish between nodes and leaves by referring to them as “domains” and “end-nodes” respectively. However they can both commonly be referred to as “nodes”. Domains can have subdomains. In Figure 4.1, the root domain has two subdomains: “/ab” and “/e”.

Each node has a label, which must be unique in that domain. The full fds address, or fds address for short, is a list of labels, describing the path from the top (the root node). While the node label is unique in its domain, the fds address is unique in the entire tree. One is a consequence of the other.

Node labels in a fds address are separated by the “/” character. Hence, the “/” character is reserved and can not be used in labels. The zero-length node label is reserved for the root node.

There is no obvious way to distinguish an end-node from a domain simply by looking at its fds address. This is similar to a file system (how do you differentiate files from directories?)
and DNS, where there is no way of distinguishing a host name (e.g. www.ecare.se) from a domain (e.g. stockholm.ecare.se) simply based on their names. If desired, an extra "/" can however be added after the FDS address of a domain to distinguish its name from that of an end-node. For example the domain "/ab" in Figure 4.1 can also be referred to as "/ab/", but adding a slash character after the end-node "/ab/ks" is not allowed since end-nodes can not have any children.

The root domain, with zero-length label, must be referred to with a trailing slash character. If not, the FDS address of the root domain is null! Hence, the root domain is always referred to as "/".

For each domain there will be a domain controller responsible for maintaining directory information for that domain. Domain controllers are logical components that can be deployed on any physical server. This is the key behind distributing the maintenance of the Feedback Tree, as will be discussed later in this chapter.

**Propagating Statistics Through the Feedback Tree**

To a computer scientist, the Feedback Tree is a tree-like data structure. Somebody within the Swedish field of medical x-ray imaging however would note from how the nodes are arranged that they correspond to real geographical and political structures. This is how the Feedback Tree is meant to be used, and although Feedback servers and their corresponding end-nodes in the Feedback Tree can carry arbitrary names, anything else than geographical and organizational references would make little sense.

Typically, one Feedback installation will be associated with one end-node in the system-global Feedback Tree. Domains will be partitioned according to regions, councils and hospitals.

A system-global statistics poll will always be initiated from the top level — the root domain. Initially, domain controllers of all top-level subdomains will be requested to compile their statistics. These domain controllers will ask their subdomains to do the same thing, and...
the requests will recursively be repeated all the way down to end-nodes. Responses will correspondingly be traversed back towards the top-level, each domain controller compiling a bulk of statistics for its subdomains and end-nodes. Eventually, the root domain controller(s) have received statistical data from all top-level subdomains. At this point, statistics are summarized at national level and the resulting data are delivered to all nodes again by recursive iteration.

In short, supplying all nodes with national-wide statistics is done in three steps: send request (from top to bottom), respond (from bottom to top) and share the results (from top to bottom).

Obviously, a domain controller can not wait indefinitely for an answer. Timeouts, although generous, must be used not to allow the failure of a single Feedback server to ruin the complete poll. These timeout must be set in accordance to the complete number of end-nodes and subdomains below in the tree. That is, a root controller must allow its top-level subdomains more time than what is given a hospital-level subdomain with one or two end-nodes.

4.3.2 The Feedback Directory Service (fDS)

The Feedback Directory Service, fDS, is the service to query and update the Feedback Tree. A distributed service, as suggested in Chapter 3, Problem statement, is indeed recommended for scalability and reliability reasons. But let us start with considering the situation of a single centralized server function, a root server, maintaining directory information for all nodes in the Feedback Tree. We will call this Alternative 1. The process of registering a node “/a/b/c” is then illustrated in Figure 4.2.

Figure 4.2 An example of registering a node “/a/b/c” on a centralized Feedback Directory Service (fDS) — Alternative 1.

Although a centralized service (i.e. not distributed), high reliability can still be achieved by replicating the server function to several physical server nodes. This is a relatively simple measure to “force” a centralized service into being distributed. However scalability is limited as every root server needs to maintain directory information for the complete set of nodes in the Feedback Tree. That is, the root server is the domain controller of all domains in the Feedback Tree.

As a measure to increase scalability, the directory server functionality could be distributed based on domains. The root server then maintains addresses to domain controlling servers, each responsible for maintaining directory information about end-nodes in their respective domains. We will call this Alternative 2.

A Feedback server registering an end-node “/a/b/c” will then first find the domain controller for “/a/b”, then register the end-node at this controller. If a domain controller does
not exist, a will register itself (at the root server) as domain controller for ”/a/b”, then register the end-node ”/a/b/c” internally. This whole process is illustrated in Figure 4.3.

![Diagram](image)

**Figure 4.3** An example of registering a node a/b/c on a partly centralized Feedback Directory Service (FDs) — Alternative II.

Clearly, this setup would increase scalability compared to the centralized setup since the root servers no longer need to maintain list of the complete set of nodes in the Feedback Tree, but only lists of the complete set of domains.

A fully distributed version of FDs will use the concept of domains and subdomains to shield the root servers from anything than top-level domains. That is, rather than binding the domain ”/a/b” to a centralized server function, the domain ”/a” will be bound to a host a, and the subdomain ”/a/b” will be registered at a rather than at the root server. The total set of data spanned by the Feedback Tree is now fully distributed in close relation to the structure of the Feedback Tree itself. This increases complexity in domain controllers since they need not only maintain a list of end-nodes belonging to that domain, but also a list of subdomains and their corresponding domain controllers. Thus, in comparison to Alternative I and Alternative II above, more advanced schemes need to be deployed to ensure data consistency.

This structure of servers with responsibilities based on domains and subdomains is very similar to how DNS or LDAP is designed. However as mentioned before, DNS as well as LDAP have a statically configured nature allowing for dynamic changes only at end-node level. The agreements on which servers are in charge of which domains are configured by humans rather than negotiated dynamically between servers. The Feedback Tree however aims at being self-configuring as far as possible.

So how would the example of registering a node ”/a/b/c” work in a fully distributed, self-configuring Feedback Tree? This is described in detail in the next section.
4.3.3 The Fully Distributed Registration Process

Figure 4.4 illustrates the registration process in a process chart. The algorithm will be described in more detail throughout this section.

Assume a new end-node on level \( n \) in the Feedback Tree. The end-node is hosted by the Feedback server \( \lambda \). There are \( n - 1 \) number of domains between the end-node and the root domain (level 0). The following high-level algorithm is used by \( \lambda \) to register the node with the distributed FDS:

**Figure 4.4** An example of registering a node a/b/c in a fully distributed, self-configuring Feedback Directory Service (FDS).
4.3 Proposed Solution

Start by assigning $i = 1$.

1. Let $i$ be the level of the current iteration and $d_i$ the associated domain.

2. Connect to the domain controller of the parent domain $d_{i-1}$.

3. Request and remember the domain controller for $d_i$.

4. If a domain controller does not exist, register $\lambda$ at the domain controller of $d_{i-1}$ as the domain controller for $d_i$.

5. If $i < n$, increase $i$ by 1 and loop from (1). If $i = n$, proceed.

6. Register the end-node with the domain controller for $d_{n-1}$.

Done.

This algorithm is illustrated in Figure 4.4, with the figure adding one extra feature: if connecting to the domain controller $d_{i-1}$ fails, corresponding to step (2) in the algorithm, the scheme illustrated in the illustration will repeatedly query the controller of $d_{i-2}$ for the controller of $d_{i-1}$. This is a small measure to increase robustness in the system, however not enough to guarantee data integrity and a completely fail-safe registering algorithm. There will be further discussions on this topic in Section 4.3.5.

The main difference between this algorithm and DNS or LDAP is step (4) in the algorithm. This ability to register domain controllers dynamically is what makes FDS self-configuring. Correspondingly, a nation-wide or even world-wide Feedback Tree can be constructed simply by providing a set of geographically distributed Feedback servers with a respective name and the address of one or several mirrored FDS root servers. No other statically configured information is needed.

The next question is whether the root servers are really necessary. The short answer is "yes", the full answer will be given in Section 4.3.5, Robustness, data integrity and efficiency.

4.3.4 The Query Process

Querying FDS is similar to querying DNS. Let us assume that a Feedback server $\lambda$, registered as node "/a/b/c", is interested in directory information (e.g. IP number) corresponding to the node "/k/1/m". If $\lambda$ is recently started, $\lambda$ will have no cached entries from former sessions. Thus, $\lambda$ starts by querying the root server about the domain controller $b$ for "/k", continues by asking $b$ about the controller $c$ of "/k/1", and finally asks $c$ for information about its domain member "/k/1/m".

All information obtained in the process, that is, the contact information for domain controllers $b$ and $c$ associated with "/k" and "/k/1", is cached by $\lambda$. Hence, unless the topology of the Feedback Tree has changed between this query and the next, $\lambda$ can contact $c$ directly for queries within the "/k/1" domain. This is very important for scalability and once again similar to how DNS works.

4.3.5 Robustness, Data Integrity and Efficiency

The algorithms on which FDS is based need to be fail-safe. Also, measures have to be taken to guarantee data integrity at every time instance. These issues will be discussed in this section.

Further, there are ways of increasing the efficiency, which will be presented last in this section.
Why Root Servers?

At the time of writing this thesis, there is much discussion in the Internet community about peer-to-peer file sharing networks such as Gnutella [9]. These networks can work without central root servers. All that is needed for a new host to join the network is a starting-point. Rather than being descendants of an authoritative root server, hosts in these peer-to-peer networks find each other by recursively asking other peers about all hosts they know of. Hosts organize themselves in a completely flat structure, i.e. not hierarchical as most other directory services — thus the name peer-to-peer networks.

The question is: can these models be used by the Feedback Tree and its rns? The answer: not easily. There are two major flaws:

1. Peer-to-peer networks are by definition flat rather than hierarchical, whereas the Feedback Tree is strictly hierarchical. The hierarchical structure of the Feedback Tree is vital for calculating and propagating statistics.

2. Given a set of n hosts running a peer-to-peer file sharing protocol, there is no guarantee that all hosts will find each other unless there is a central starting point. Without a root server, it is likely that hosts form themselves in several smaller, separate networks rather a single large network. The composition of the networks will be determined by the starting points given to new hosts.

Thus, even with today’s emerging peer-to-peer directory service technologies, root servers are necessary for building the Feedback Tree.

Fail-Safe Algorithms and Data Integrity

The following mechanisms are used to guarantee fail-safe algorithms and data integrity at every time instance:

1. End-nodes and subdomains will send a “I am alive” message to their parents (i.e. their domain controllers) with a regular time interval t.

2. A domain-controller that has not heard from an end-node or subdomain for a time of 2t will remove that node from its list of domain members (i.e. remove the node from the Feedback Tree).

3. A node that sends a “I am alive” message that is rejected by its parent will try to re-register itself following the standard registration procedure.

4. If encountering an unrecoverable error in the registration process, i.e. a domain controller suddenly disappears from the network, the registration process will be restarted.

5. If encountering an error when finding or communication with a node, the node will be purged from all caches and a new query will be initiated to find the node on the Feedback Tree.

6. All rns messaging over the network will have a timeout value, after which an error will be thrown. A node must not wait indefinitely for an answer.

7. Aggressive locking of domains will be used in absence of the more sophisticated alternative of distributed transactions. A domain controller can, on request, mutually exclusively lock a subdomain. That is, read and write access on that domain is exclusively granted to the owner of the lock.
4.3 Proposed Solution

Only the domain controller of the domain in question is allowed to lock the domain. The lock will not be granted indefinitely but rather time-out after a given time interval. Examples of the use of locking will be presented later.

Items (1)–(3) in the list ensures that the Feedback Tree will eventually converge to being fully valid even if nodes leave the network, change address or crash without warning.

Efficiency

1. The root server function can be duplicated to several physical hosts. Typically, as in DNS, one host will act primary server and the others will mirror the database of the primary server. This increases efficiency as well as scalability but once again requires mechanisms to ensure data consistency between mirroring servers and the primary server.

If full consistency is to be guaranteed at every time instance, that is, all mirroring servers will *always* be guaranteed to have an up-to-date copy of the root server’s database, the primary server needs to block read access from clients until it is certain that updates have been propagated to all mirrors. This is safe but can introduce considerable delay. The DNS uses a somewhat more reasonable approach, where inconsistencies are accepted for as long as it takes to push the updates to the mirroring servers. It is suggested to use this approach also in FDS. After all, clients will repeat their query if the information they get (i.e. an address is invalid) is out-of-date.

Updates are only allowed at the primary root server. So what happens if the primary root server goes down? There are two obvious solutions to this problem. Either updates are not allowed during the time the primary server is not available, or one of the mirroring servers will take over the update responsibility. Then, of course, the original primary server needs to be updated when it comes back to life.

Generally, duplicating a read and write-enabled server function such as the Feedback Tree root server is a non-trivial task. Details of this will be given no further attention in this thesis. As will be discussed later, the root server functionality can be replicated with or without the needs for eCare to implement everything from scratch. Namely, if choosing to rely on existing technologies such as for example CORBA Naming Service, the ability to replicate servers sometimes comes with the implementation.

2. Nodes will *cache* all information discovered through queries. Hence, unless the topology of the Feedback Tree has changed, nodes can use their cached information to contact end-nodes or domain controllers directly without repeatedly having to find them through DNS queries. Item (5) in the list above ensures that caching of outdated information does not cause ill-effects on the system.

3. In the fully distributed registration process (described in Section 4.3.3), a node will register itself as domain controller for a domain in case a domain controller does not already exist. Later, other Feedback servers may come and register end-nodes and/or subdomains to that domain. It is not always the case however that the node being “first” at that domain, thus being registered as domain controller, is the best suited candidate to run the domain controller in the long run. Details of the negotiation process will be described in the next section.

---

1Why? Because if given old information, for example an outdated IP address to a domain controller, the client will fail querying that node. Either the node will not respond at all, or the server will reject the messages that are targeted at a node address the server is not familiar with. Thus, according to item (5) on the numbered list ensuring fail-safe behaviour, the query will be restarted.
4.3.6 Domain Controller Negotiation

Ideally, the node responsible for maintaining directory information and calculating statistics should be the node least loaded, with the most processing power and the most network bandwidth. Correspondingly, the domain controlling role will periodically be negotiated between Feedback servers that have end-nodes or subdomains in that domain.

The domain controller initiates the negotiation process. Firstly, the domain controller locks its domain at its parent. All hosts that have registered children (end-nodes or subdomains) to the domain are then asked for their domain weight, a quantity $w$ representing that host’s suitability to control the domain. Possible parameters that might influence the value of $w$ are system speed, current and past load, network bandwidth, uptime, etc.

If the domain controller finds a host with considerably higher weight than itself, a transfer of the domain controlling authority is performed. The exact details of this transaction is beyond the scope of this thesis. Instead, focus will now be shifted to describing the high-level services that must be included in fds in order to accomplish what has been discussed in this chapter so far.

4.3.7 A High-Level fds Protocol

Function: authenticate

options: None.

parameters: None.

description: Initiates handshaking and have the two communicating hosts authenticate each other. Preferably using x.509 certificates.

Function: list

options: end-nodes | subdomains

parameters: <fds address>

description: Requests a list of end-nodes or subdomains under the given domain. Only a valid request at domain controllers.

Function: get

option: hostweight

parameters: None.

description: Gets the domain controller weight $-1 \leq w \leq 100$ of the host. A value of $-1$ means the server will under no circumstances accept a domain controller role.

option: <key>

parameters: <fds address>

description: Gets directory information based on the tuple {key, fds address}. Example: "get ipnumber /ab/ks".
Function: update
option: <key>
parameters: <fds address> <value>

**description:** Updates directory information based on the triple \([key, fds address, value]\). Example: "update ipnumber /ab/ks 130.236.1.27".

Function: register
option: end-node
parameters: <fds address>

**description:** Registers an end-node with the given address. The end-node will be bound to the ip-number of the host issuing the request. This function call is only valid at the domain controller of the end-node’s domain. Example: "register end-node /ab/ks".

option: subdomain
parameters: <fds address>

**description:** Registers a subdomain with the given address. The subdomain will be bound to the ip-number of the host issuing the request. This function call is only valid at the domain controller of the subdomain’s parent domain. Example: "register subdomain /ab".

Function: lock
options: None.
parameters: <fds address>

**description:** Locks the domain with the given fds address. The request is sent to the domain controller of the parent domain. Only domain controllers are allowed to lock their respective domains. Example: "lock /ab". This will cause the domain controller for "/" to deny requests to hosts requesting any directory information about "/ab". Typically, this will be used to stop hosts trying to register end-nodes or subdomains to a domain where domain controller negotiation is taking place.

Function: unlock
options: None.
parameters: <fds address>

**description:** Unlocks the domain with the given fds address. The request is sent to the domain controller of the parent domain. Only the host that acquired the lock is allowed to unlock the domain.
Function: still_alive

options: None.

parameters: `<fds address>`

description: This function is used by a Feedback server hosting a `fds` address (the parameter) to update the domain controller of the parent address of its existence. If these messages fail to arrive, nodes will be purged from the tree. Example: the host serving the end-node `/ab/ks` will periodically use `still_alive /ab/ks` at the domain controller of `/ab`.

Function: request_statistics

options: None.

parameters: None.

description: Requests the targeted end-node or subdomain to reply with a compiled bundle of statistics.

Missing Functions

All functions for delegating domain authority have deliberately been left-out as more detailed discussions on this topic is beyond the scope of this thesis.

4.3.8 Choosing Implementation Protocol

Finally, there is the question of choosing a suitable technology platform to implement the Feedback Tree and the `fds`. The options that still stand are CORBA Naming Service and a proprietary solution. These two candidates will be discussed in more detail before a final proposal will be made at the end of this section.

Candidate 1: CORBA Naming Service

Here, the Feedback Tree will be represented using a federated naming graph. Domain controlling Feedback servers are hosting the naming context objects to which the nodes are bound. End-nodes are represented using a custom node object class and subdomain are represented with new naming context objects. The naming contexts will maintain as little directory information as possible. Consequently, end-nodes will be bound to the contexts with a corresponding node label but all other information, for example node description, is retrieved from the end-node object rather than the domain controller (the naming context).

To each naming context, a statistics engine object is also bound. This object is responsible for forwarding poll requests to subdomains and end-nodes, and compiling responses before replying upwards (as described in Section 4.3.1).

Querying a federated naming graph is straightforward. The developer can take full advantage of CORBA’s location-transparent object models and consequently traverse the tree as if it was all based on the local machine. The `fds” get” function is implemented with calls to naming context objects (e.g. `get subdomains`), and direct calls to end-node objects in some cases (e.g. `get description /ab/ks`).
4.3 Proposed Solution

Registering nodes are similarly straightforward. The “register” and “update” calls are mapped to naming context objects (e.g. `register end-node /ab/ks`), or to end-node objects (e.g. `update description /ab/ks "Karolinska Sjukhuset"`).

The “I am alive” signalling, and automatic removal of “lost” nodes, is something that needs to be developed on top of the CORBA Naming Service.

Caching of already performed node look-ups can be achieved by keeping copies of all object references that are discovered when traversing the Feedback Tree. For example a hashtable can be used to store remote object references with their respective RDS addresses used as hashtable keys.

The suggested CORBA Naming Service implementation distinguishes between two separate domain controlling functions: maintaining directory information (via a naming context object) and compiling statistics (via a statistics engine object). Out of these two, managing statistics is expected to be the most computing-intensive, and probably the most important component to negotiate between servers. It is also the simpler service to move around the network. All it takes is that a binding to a new statistics engine object, hosted at a different server, replaces the old binding at the domain’s corresponding naming context object. This is a one-call operation and distributed transactions are not be necessary.

The naming contexts themselves are more difficult to shuffle around between servers. It is possible, but not encouraged by the way CORBA Naming Service is designed. The question is whether this is really necessary, or if negotiating compilation of statistics is enough. This is not a question this thesis will attempt to answer but will rather have to be evaluated by eCare based on the real deployment environment, should this candidate be chosen.

Automatic mirroring of naming contexts, used to replicate servers, is supported by some CORBA Naming Service implementations, including Borland’s ubiquitous VisiBroker [64] and Prism Technologies’ OpenFusion CORBA services [74]. Leaving the non-trivial task of replication to the infrastructure rather than having to do it yourself is obviously an enormous advantage.

There are many pros, but also two cons with the CORBA-based approach described in this section. Firstly, a fail-safe CORBA Naming Service implementation costs money. While writing this thesis, some effort has been put into finding a free or open-source initiative, but with no success. The VisiBroker product then lies close at hand, which is roughly priced at 20,000 SEK per CPU on the machine it is deployed on. This may sound costly, but one has to remember that only the root servers need to be replicated — if other nodes fail, the Feedback Tree will recover automatically. Thus, for all nodes but the root nodes, the basic (and free) ORB supplied with the Java platform, Java idl, may be enough.

Then, there is the question of CORBA and firewalls. Commercial products like for example VisiBroker can generally overcome any potential problems. Java idl from Sun Microsystems however is not a mature product. The included ORB is compliant with some, but not all, of CORBA version 2.3. This is not satisfactory when current CORBA version is 2.4 and version 3.0 is just around the corner.

Further, Sun Microsystems state in their own Java idl documentation:

> “Although it is true in theory that ORBs written in different languages should be able to talk to each other, we haven’t tested the interoperability of the Java ORB with other vendor’s ORBs.”

OMG has stated that the issues surrounding i1op and firewalls will be fully resolved in CORBA version 3.0, but the question is if and when Java idl will fully adopt this standard. Meanwhile, developers have to rely on commercial products and their respective solutions to the problem.

---

2That is, a service implementation that supports replication and load-balancing transparently.
If Java idl is to be used on all servers but the root servers, it first has to be evaluated whether Java idl:

1. ...is compliant with corba candidates for the root servers (e.g. Borland VisiBroker or Oracle Application Server).
2. ...encompasses all functionality needed to fulfill its tasks.
3. ...works through hospital firewalls.

Java version 1.4 is around the corner, due to be released in its final form early 2002. With this release Sun Microsystems have made enhancements to Java idl, today (December 2001) documented in early draft documents. Consequently, exact details are still unclear. When 1.4 is officially released, it should be evaluated according to the above three items.

Candidate 2: a Proprietary Implementation

It is not difficult to model data structures for the Feedback Tree using object-oriented paradigms. Correspondingly, Java is a highly suitable language for this task. But how is the information communicated between servers? All high-level rds functions described in 4.3.7 needs to be communicated over the network, and mechanisms for domain controller negotiation and possibly root server mirroring need to be designed and implemented.

For a completely proprietary directory service, eCare has mainly two options at hand: a Java rmi-based solution, or a custom low-level protocol.

Let us begin by considering the former: Java rmi as a message carrier. Then, Java classes representing domain controllers and end-nodes are constructed from scratch and shared via Java rmi. Similarly to corba, Java rmi provides location-transparency and the Feedback Tree can be traversed as if all objects were local.

Generally, a rmi-based solution will be quite similar to the corba-based approach. However Java rmi provides no support for load-balancing or server replication, and the mirroring of root servers would need to be developed from scratch. This is far from trivial, and consequently, a Java rmi-based solution would at least initially have to stand without a fail-safe root server function. As a more long-term solution, j2ee provides distributed transactional capabilities that would aid in developing such features.

Java rmi is not language-independent, although rmi can be used over iiop to provide limited corba connectivity. Binding to Java is not an obvious drawback at this stage, but may be in the future if eCare partly or fully migrates from the Java platform.

On the positive side, there are no problems having Java rmi work through firewalls.

The second option, a completely custom communication protocol, is taking a step away from any form of middleware. Everything then has to be developed from scratch, including data marshalling, message parsing, managing tcp connections, etc. This allows for a protocol highly optimized for its specific tasks, and complete language-independency for as long as protocol adapters can be developed in that language.

A custom low-level protocol is probably how things would have been done some 15 or 20 years ago. The main reason, of course, was that there did not exist much middleware to rely on. Although still possible today, not many developers choose to ignore the existence of supporting middleware. Middleware products often cost money, but so does keeping software developers busy with the basic nuts and bolts of distributed computing rather than having them focus on the problems their product is set to resolve.

Correspondingly, a custom low-level protocol can not be recommended unless there are very specific reasons and all other options have been ruled out.
4.3 Proposed Solution

4.3.9 Final Proposal

Short-Term

corba is not a suitable candidate for quick short-term solutions due to possible deployment problems in combination with firewalls.

A Java RMI-based solution without mirroring of root servers or full negotiation of domain controlling authority is something that can be done relatively quickly. This is thus the recommended short-term solution.

Long-Term

Eventually, as the Feedback Tree grows, redundant and fail-safe root server functionality will be important. corba certainly delivers interesting features of this kind, is completely language-neutral, and is not too different from Java RMI. Migrating from a custom-made Java RMI-based directory service to using corba Naming Service does not necessarily have to be a great effort.

With corba version 3.0, the issues surrounding corba and firewalls will hopefully be resolved.

It is recommended that eCare keeps a close eye at the orb market, and evaluates the corba capabilities of the upcoming Java platform version 1.4. When time is ready, it is then suggested that the Feedback Tree is implemented using a combination of J2EE and corba technologies. Then, the non-trivial tasks of maintaining data-integrity throughout server replications can be left to the infrastructure, and eCare’s software engineers are able to focus on more high-level concepts.
Chapter 5

Proposed Solution: Data Propagation

Based on the needs and requirements presented in previous chapters, three distinct classes of data propagation can be identified:

1. Feedback ⇔ Feedback (statistics and diagnosis discrepancies)
2. rIS → Feedback (diagnosis data)
3. PACS → Feedback (image data)

Class 3 will not be further discussed as it is beyond the scope of this thesis. The other two classes will be discussed throughout the rest of this chapter.

A short-term and long-term recommendation for Feedback–Feedback communication is concluded at the end of Section 5.1. Section 5.2 is devoted to a novel application introduced in this thesis: InfoBroker. This application is tailored to facilitate Feedback–rIS and Feedback–PACS connectivity, although specific implementation of Feedback–PACS connectivity is beyond the scope of this thesis.

5.1 Feedback ⇔ Feedback

As stated before, current Feedback implementations rely on Java rmi as middleware. There are no current plans within eCare to migrate from this platform, and certainly not to stop using Java as the central implementation language.

Internally, Java rmi uses its rmi Transport Protocol [71] for network traffic. The rmi Transport Protocol makes use of two other protocols: Java Object Serialization and possibly http. The Object Serialization protocol is used to marshal call and return data, and http for transport over the network when circumstances warrant. Namely, http can be used to invoke remote methods through a firewall that does not allow for generic tcp streams. When http is not used, rmi Transport Protocol uses one of its StreamProtocol or MultiplexProtocol for transport. The former, which is default behaviour, simply pushes serialized (marshalled) objects through a tcp socket stream. The purpose of the multiplexing protocol is to allow a client to connect to a rmi server object in some situations where that is otherwise not possible. For example, some security managers for applet environments disallow the creation of server sockets to listen for incoming connections, thereby preventing such applets to export rmi
objects and service remote calls from direct TCP socket connections [71]. For details on the MultiplexProtocol, see [71].

The data that needs to be propagated between Feedback servers — statistics and diagnosis discrepancies — are represented as programming objects within the application. After being transferred over a network, they need to be re-assembled (unmarshaled) into their original form to be used by the Feedback server at the other end. Several different approaches to this chain marshal-transfer-unmarshal can be taken, as will be discussed throughout the following sections.

5.1.1 Candidate 1: mom-based Solution

Description

Object marshalling is done using either xml-encoding or Object Serialization, and JMS (Java Message Service) is used to pass the data to the mom server for transport.

The mom server will then handle data transfers using its own algorithms and protocols, and at the other end, data will be fed to Feedback once again via JMS and will be re-assembled in Java.

The mom also handles application-level messaging, such as requests for data, error messages, etc.

Reliability & Efficiency

Mom servers are built primarily for reliable data transfer, and using a well-known mom product to shuffle data around the network is perhaps the most reliable way of doing it to date. Efficiency in data transfer will differ between mom products since they tend to use custom protocols.

XML-encoding, and particularly decoding can be very expensive. Java’s built-in Object Serialization is less expensive, although critics accuse Java in general and and Java Object Serialization in particular of being inadequate for high-performance computing [8]. The fact remains that execution of platform-independent Java bytecode will probably never reach the speeds of compiled C/C++ or Fortran, but the the programmer can at least implement her own optimized serialization and unserialization (marshaling/unmarshaling) for particular Java classes, should the need occur. Further discussions on XML versus Object Serialization will follow below.

Firewall Tolerance

Many mom products are designed more for a corporate network than the Internet, and as a result do not easily operate across institutional firewalls. There are workarounds however. The ibm mqseries product, for example, can work through firewalls using HTTP as data carrier.

Security

Mom products such as for example ibm mqseries are extensively used in for example banking and financial applications. With the right mom product, eCare will be able to build a system as secure as anything on the public market to date.
Advantages & Disadvantages

With MOM-based data transfer models, and XML as transport format, coupling is kept at a minimum. There is however a restriction. The functionality provided by MOM products is often achieved through a private lower-level protocol that is owned and controlled by a single company, and which is often difficult if not impossible for other vendors to connect to. Thus, mixing MOM products is not always possible. But then again, using standard JMS to access the MOM layer from Java un-binds the Java implementation from a specific MOM product. If so, the underlying MOM product can be replaced without ripple effects on the Java code.

XML is platform and language independent, but also introduces other features. It is, for example, possible to define simple transformations for automatic conversion into any given form, e.g. HTML or PDF.

Using Java’s Object Serialization rather than XML means binding to the Java language. In such a setup, full use is still made of the flexible data transfer models of MOM, however, no application other than a Java application will be able to understand the data.

As often is the case, increased flexibility leads to increased complexity and thus increased needs of processing power. Off-the-shelf XML parsers are available for most programming languages, including Java\(^1\), but although using them requires little effort, they do some serious formatting and parsing under the hood and are correspondingly expensive to use.

Also, MOM implementations cost money. Since there is no obvious need for a MOM-based middleware product when it comes to Feedback directory services, the question is whether the (significant) financial cost of a MOM middleware can be motivated.

5.1.2 Candidate 2: XML and a File Transfer Protocol

Description

Object marshalling is done using Java’s built-in XML encoding, and the data is propagated over the network using a file transfer protocol such as FTP, SSH or HTTP. The transfers can be initiated from Java, or be run as separate “jobs” transferring files that Feedback prepares for example in a directory on the local filesystem.

A middleware such as for example Java RMI is used for application-level messaging.

Reliability & Efficiency

XML encoding and decoding can, as mentioned above, be very expensive.

File transfer protocols are often optimized for efficiency rather than reliability. None of the protocols FTP, SSH or HTTP are tolerant to network deficiencies and the programmer will need to ensure by herself that files (XML documents) are re-transmitted upon errors.

Firewall Tolerance

HTTP is the transport layer over which the vast majority of Internet messages are conveyed today. Thus, it can leap most corporate firewalls. Making FTP work through firewalls can be a problem. Either the firewall needs to support FTP explicitly by using a proxy server to relay connections, or FTP must to be used in “passive” mode, which is not supported by all FTP servers.

\(^1\) A publicly available standard extension for XML-related parsing and formatting, Java API for XML Processing (JAXP), has been around since 1999. Starting with Java platform version 1.4, due to be released in early 2002, this extension is an integrated part of the platform.
ssh is, similarly to HTTP but differently from FTP, based on a single TCP connection that can easily be administered through a firewall. For more information on HTTP, FTP, SSH and firewalls, see for example [53].

**Security**

HTTP was not originally designed to be a secure protocol, however HTTP is often used over TLS or SSL sockets and is then referred to as HTTPS.

The FTP protocol poses a serious security problem: it sends usernames and passwords unencrypted across the network. Thus, it is trivially easy for hostile parties to capture ("sniff") those usernames and passwords, and then remotely login to the system as if they were your authorised users (or application components).

ssh uses public key cryptography for connection and authentication, similarly to TLS and SSL. The set of encryption ciphers are also similar, including standards such as DES and 3DES. ssh is widely used by network administrators and must be considered a safe protocol.

**Advantages & Disadvantages**

Using XML over a file transfer protocol rather than a commercial MOM product still leaves room for the flexible and platform-neutral nature of XML, however everything that MOM delivers in terms of robustness and reliable data transfers must be implemented by the application developer himself.

Also, how to use ssh in combination with Java is not obvious. Most likely, third party software will need to be used and called from Java. Using FTP and HTTP in Java is trivial since the Java platform supports these protocols at low-level and the API includes all functionality needed.

**5.1.3 Candidate 3: CORBA-based Solution**

**Description**

Object marshalling is done by the CORBA implementation (e.g. Borland VisiBroker [64] or JacORB and Prism Technologies’ OpenFusion CORBA services [74]) and data transfer will be handled by the corresponding CORBA Event Service [33]. The ORB implementation carries data across the network using the IOP protocol.

**Reliability & Efficiency**

As mentioned in Section 2.4.9, Chapter 2, there has been a long, still on-going debate in the industry as to whether CORBA is suitable for data transfers or not. Benchmarks often show on considerable overhead for making operation calls (equivalent to method calls in Java) and marshalling objects through an ORB. This should not come as a surprise, as CORBA is extremely general and flexible [18]. Increased flexibility almost without exceptions comes with increased complexity.

One has to remember however that performance will depend largely on the ORB implementation being used. Some products are faster, more scalable and more suitable for large-scale systems than others, and high-performing, real-time systems have been built using CORBA [24, 35].
5.1 Feedback

Firewall Tolerance

Chapter 2 stated that the combination of firewalls and CORBA can be problematic, however dependant on the CORBA implementation being used.

Security

HTTP can be used over TLS and SSL sockets, taking advantage of the increased security these protocols deliver. Also, CORBA contains strong security mechanisms at application level, making it possible to set fine-grained access levels on objects and their operations.

Advantages & Disadvantages

Obvious advantages of a CORBA-based solution would be many of the advantages generally associated with CORBA — platform and language independence, loose coupling, and the richness of the CORBA services, including features such as distributed transactions and asynchronous messaging.

5.1.4 Candidate 4: Java RMI

Description

Object marshalling is done via Java Object Serialization or XML encoding/decoding. Data transfer is done using RMI Transport Protocol.

Reliability & Efficiency

No guaranteed delivery of data, as in the case of MOM or CORBA messaging, is provided by Java RMI. Thus, the programmer needs to handle transfer errors by herself when they occur.

The RMI transport protocol is minimalistic, thus efficient. When forced to use HTTP as lower-level transport, some overhead is introduced as a proxy server needs to be set up that listens to a HTTP port and encodes/decodes all RMI messages to and from HTTP messages. This is similar to what the programmer would need to implement herself should HTTP be used as transport in favour of Java RMI.

Java Object Serialization is less expensive than both XML encoding and CORBA object marshalling, as has already been discussed.

Firewall Tolerance

Using HTTP for transport, Java RMI and its RMI Transport Protocol works through just about any firewall.

Security

Java RMI can be used in combination with TLS or SSL for secure authentication and encryption of data.

Advantages & Disadvantages

Using RMI is convenient since existing Feedback is based largely on this standard. Thus, extending the connectivity from within-application to also between-application need not be
a dramatic move. Java RMI also comes without any charges whatsoever since it is an integral part of the standard Java platform.

A Java RMI-based solution is however the only alternative out of the four candidates presented that is not language-neutral. Luckily, migrating from Java RMI to CORBA is not too painful should the need for a language-neutral environment arise.

5.1.5 Final Proposal

Short-Term

Candidate 1, MOM, is quickly ruled out due to its financial costs and the negative effects of having to rely on yet another software vendor. Candidate 3, a CORBA-based transfer model, is taking a risk considering the potential problems associated with hospital firewalls and nor, and should be avoided until a careful market research of the ORA market, focused on reliability and firewall tolerance, has been made.

For a short-term solution, that leaves us with candidate 2 and 4 ― using either XML or Java Object Serialization for object marshalling and then Java RMI or a file transfer protocol for data transfer.

FTP is quickly ruled out as data carrier due to weak security models and possible deployment problems in combination with the multiple layers of firewalls that must be expected. That leaves SSH, HTTP and Java RMI as candidates. The SSH protocol can then be ruled out as it adds no or minor security advantages compared to running HTTP over TLS or SSL, but adds complexity by not being an integrated part of the Java API. Of the remaining two, HTTP and Java RMI, the former adds language neutrality whereas Java RMI binds to the Java language. On the other hand, Java RMI is a much more convenient choice, already containing the functionality (e.g. addressing, message preparation, etc) that would otherwise need to be developed by the eCare themselves.

The increased coupling that comes with binding to Java is obviously not a problem until eCare partly or fully migrates to another language.

So, the most reasonable short-term alternative for data transfer, and thus the suggested solution, is Java RMI. Next, there is the question of what form the data shall be marshalled into: language-neutral XML or Java-specific binary streams produced by Java Object Serialization? It is clear that the latter would be the most convenient, since the objects already exist in the application and very easily can be piped through Object Serialization². In fact, calling a remote method through Java RMI, passing an object as a parameter, the object will be serialized and de-serialized at the other end completely transparent to the developer. So why bother with XML?

There are several reasons why XML might be a reasonable alternative:

1. Consider the alternative: a Java-specific transport model (Java RMI) transporting data that is also Java-specific. Although Java by itself is platform-neutral, binding to a specific language is something that does not conform to the general idea of flexible and loosely coupled application design.

2. Using XML, the Java classes representing the programming objects can be modified without compatibility problems between servers and versions, given that the XML syntax used to represent objects (the DTD) remains the same. And if needed, implementing

² Interestingly, in the upcoming version 1.4 of the Java platform (due to be released early 2002) Sun has implemented XML encoding classes that automatically generate XML based textual representations of any object. Decoding classes can later parse the XML back into Java objects. This is promising but does not offer control over the structure and syntax of the XML files, which means eCare would still want to develop their own encoding and decoding schemes.


support for different versions of XML documents is simple — Java RMI and Java Object Serialization on the other hand does not support multiple versions. The hosts at each end of the network must always have access to an identical set of the class files describing the objects that are being marshaled and unmarshaled.

3. XML is human-readable, which eases debugging.

4. XML introduces many other features and possibilities, for example transformations into HTML or PDF.

On the other hand, XML is not well suited for binary data such as images. An arbitrary stream of bits cannot be plunked into a document, because all the bits in a XML document must be legal characters in legal syntax in the same character encoding as the rest of the document. However, it is possible to encode binary data into characters, and then put the result into an XML document. This binary-to-text conversion will result in larger files, with the Base-64-encoding [15] used in for example binary attachments in e-mails as a worst-case scenario, enlarging the data with \( \frac{1}{4} \). Better suited binary-to-text encoding schemes, combined with compression algorithms, can however produce text files similar in size to the original binaries.

It has already been stated that XML encoding and decoding is expensive in comparison to most other alternatives. The question is however what real impact this will have on the systems. Almost certainly, other factors such as for example network bandwidth will ultimately be the limiting factors.

Thus, the proposed short-term solution for Feedback–Feedback communication stands clear: Java RMI for transport and XML for encoding of data.

Long-Term

In a longer perspective, binding to the Java language is not ideal. Gradually migrating to a mature, cross-language middleware technology such as CORBA is recommended. CORBA and XML live in perfect harmony and there will be no obvious need to change the data marshalling schemes.

Where does J2EE (Java 2 Platform, Enterprise Edition) come into all this? Chapter 2 stated that J2EE was a strong candidate when building distributed application. The strength, however, are in within-application rather than between-applications connectivity. For the latter, J2EE relies on other supporting technologies such as for example MOM or CORBA. Also recall from Chapter 2 that J2EE actually uses Java RMI for most of its tasks, thus, introduces little new when it comes to raw data transfer.

5.2 **ris and pacs → Feedback: InfoBroker**

For the next level of connectivity, enabling Feedback to integrate with ris and pacs, this thesis introduces a new application: InfoBroker. The work includes system design as well as implementation, this report however will focus almost exclusively on high-level application design.

InfoBroker enables ris and pacs connectivity. Specifically, InfoBroker assists Feedback in extracting relevant data from these systems, isolating Feedback servers from external networks and providing another abstraction layer between heterogenous ris and pacs implementations and the Feedback. Later, InfoBroker may very well be extended to handle other types of information flow, for example providing a broker for external applications to extract data from Feedback rather than simply the other way around.
InfoBroker deploys in a fully distributed environment, thus, must comply with requirements and "good" distributed design guidelines as discussed in Chapter 2. Specifically, as listed in Chapter 2, InfoBroker needs to comply with the following list of items:

- All data being sent on a vulnerable channel (e.g. streamed over Internet) is encrypted to prevent eavesdropping and manipulation of data.
- A strong authentication scheme is established to ensure that the identity of users as well as system components can be reliably ascertained.
- The facility exist to log every operation if desired.

Further, the following must be considered:

- Minimum level of application-internal coupling
- Minimum level of coupling in external interfaces
- Scalability and efficiency
- Reliability and fault tolerance

All the above items will be discussed in the following sections.

5.2.1 Architectural Overview

The chosen platform for InfoBroker is Java. There are several reasons for this:

Flexibility in deployment
The cross-platform aspects of Java are valuable as they enable InfoBroker to be binary compatible across platforms, including most UNIX flavours as well as Microsoft Windows and Apple Mac OS X. The main pitfall of Java cross-platform deployment is graphical user interfaces (GUI), in particular specific features such as drag-and-drop. However as InfoBroker does not contain any GUI, this is not a problem. And if a GUI is to be designed, for example an administrative client, it will be designed using a client-server approach, independent on the implementation language of InfoBroker.

Fast development and rapid prototyping
Enabled by for example the richness of the Java API, Java’s close integration to Internet, and Java’s security models and error handling.

Conformance to eCare’s other development platforms
Implementing InfoBroker using the same language platform as Feedback and eCare’s other products enable reuse of sourcecode and makes InfoBroker fit neatly in eCare’s overall software development models.

Please refer to Figure 5.1 for an overview of the design of the InfoBroker application. The inner core of the application, InfoBroker, represents a Java class that, with help of a hierarchy of supporting classes, provides most of the application. Most notably, these classes include a set of RIS and PACS agents, tailored to communicate with vendor-specific RIS and PACS. The RIS and PACS routers are responsible for delivering requests and responses to and from agents, each agent responsible for communicating with one RIS or PACS instance.

An outer layer, InfoBroker Service, provides a network service based on the methods in the InfoBroker class. This enables Feedback to access InfoBroker’s functions via the network, e.g. request a diagnosis from a RIS or an image from a PACS.
Figure 5.1 A design view of InfoBroker.
The middleware used to communicate this service is Java RMI. The three main reasons for this choice are cost-efficiency, compliance with the rest of the Feedback platform, and a tight time schedule. If time would be less of a problem, a more generic CORBA based solution would be preferred since the language-independent nature of CORBA introduces less coupling than the strictly Java-based Java RMI. However since InfoBroker and Feedback are both written in Java, binding to Java is not a problem at this time.

An outer layer to InfoBroker Service, the InfoBroker server, represents the runtime environment with a number of running threads. The InfoBroker and its associated service runs in one. The other threads are:

**LogMailer thread**
Compresses (zip compression) and e-mails all InfoBroker log files to one or several administrators, with a given interval.

**ErrorMailer thread(s)**
E-mails reports of RIS or PACS data causing error or confusion, more on this in section **Fault tolerance** (Section 5.2.2). There will be one separate ErrorMailer thread associated with every RIS agent.

**Polling thread(s)**
There will be one polling thread for each RIS agent. Each polling thread periodically queries the RIS, via the polling thread’s associated RIS agent, retrieving all data since last poll. This data is then pushed into Feedback via the Feedback→InfoBroker RMI interface.

**Retry thread(s)**
There will be one retry thread for each RIS agent. The retry threads periodically retries to process the RIS records that have previously been marked as bad. More on this in section **Fault tolerance** (Section 5.2.2).

### 5.2.2 InfoBroker (RIS) Fault Tolerance

In the process of fetching a number of records from a RIS and pushing them into Feedback, several different types of errors can be occur. Please see Table 5.1 for an overview.

InfoBroker maintains two types of lists — bad and rejected — with references to RIS records that have caused error(s) in the system. There will be one set of lists for each active RIS agent.

InfoBroker does not assume that the RIS nor Feedback will be able to tell exactly what went wrong, and at what position in the stream of records. Making such an assumption would make InfoBroker vulnerable to bugs in RIS implementations and Feedback, and at network level. Instead, upon error, InfoBroker uses an algorithm similar to binary search to iteratively narrow the interval of diagnosis data until all diagnoses causing errors are found and marked.

The algorithm is straightforward: upon one or several errors, whether they are database errors, bad data errors or rejected data errors, the interval $n$ is split in two equally sized parts. Next, the procedure recursively calls itself on the two parts, and the whole process is repeated until an end-criteria is met.

This recursive loop has two endpoints:

1. A chunk of $n$ records are read from RIS and fed into Feedback without errors.
2. The interval has been narrowed down to $n = 1$, and errors still occur, at which the record is marked as bad or rejected depending on the type of error.
### Table 5.1 InfoBroker errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
<th>Immediate measure taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database error</td>
<td>No or corrupt data being returned from database.</td>
<td>InfoBroker will internally mark the affected records as bad.</td>
</tr>
<tr>
<td>Bad data error</td>
<td>InfoBroker does not understand or accept the data returned from the database. For example, a diagnosis might lack a valid doctor’s signature.</td>
<td>InfoBroker will internally mark the affected records as bad.</td>
</tr>
<tr>
<td>Internal error</td>
<td>InfoBroker raises an unknown exception internally when handling the data. Possibly an InfoBroker implementation bug.</td>
<td>None.</td>
</tr>
<tr>
<td>Feedback error</td>
<td>Feedback is not responding properly.</td>
<td>None.</td>
</tr>
<tr>
<td>Rejected data error</td>
<td>Feedback functions normally but does not accept the data.</td>
<td>InfoBroker will internally marks the affected records as rejected.</td>
</tr>
</tbody>
</table>

Table 5.1 InfoBroker errors

Please refer to Figure 5.2 for an illustrative example. At the first iteration, a chunk of 8 records is processed (fetched from ris, fed into Feedback) at which an error occurs. InfoBroker does not necessarily know where. The interval is split in two parts, each new part having \( n = 4 \), and new attempts are made on the two parts respectively. The lower part is successfully processed on the first iteration. However processing the upper part once again leads to errors. Thus, the interval is once again split in two, each part now having \( n = 2 \). The lower part requires three more iterations to complete, and the upper part completes on the first iteration.

In total, the recursive loop has iterated 7 times. Generally, the number of iterations needed will depend on three factors:

- the size of \( n_0 \)
- the number of errors \( e \), where \( 0 \leq e \leq n_0 \)
- the distribution of the errors

How can the distribution of errors affect the number of iterations needed? The short answer is that InfoBroker, unlike binary search, accepts interval lengths \( n \) that are not powers of two\(^3\).

Given \( n \) is a (positive) power of two, it is obviously possible to split \( n \) in two equally sized chunks of size \( \frac{n}{2} \), which in turn can be split in two all the way down to parts of length \( n = 2^0 \). Repeatedly dividing any other \( n \) (not power of two) by two however will eventually result in an odd valued \( n \), which in the following iteration will be split into two parts of different length. For example \( n = 5 \) will be split in two integer parts: \( n = 2 \) and \( n = 3 \). In this example, a single error in the interval \( n = 5 \) would imply two more iterations if the error originates from one of the first two records and two or three iterations if the error originates from one of the last three.

Simply speaking, in a worst-case-scenario distribution of errors, the number of iterations needed to track down the error(s) will equal the number of iterations needed to find the same number of errors in a larger \( n \) equaling the closest greater power of two.

---

\(^3\)If there are, say, 23 items, binary search numbers them 0–22 and adds 23–31 as dummy keys.
Before the error-tracking algorithm can be analyzed any further we need to answer the following question: how many times can you divide a number \( N \) in half before it will reach 1?

**Proof** For every number \( N \) there is an integer \( K \) such that

\[
2^K \leq N < 2^{K+1}
\]  

(5.1)

Assume a loop dividing \( N \) by two, replacing \( N \) with the new result. When we divide \( 2^K \) by two \( K-1 \) number of times it will equal two, and when we divide it once more, it will become one. Therefore the loop will repeat \( K \) times when \( N = 2^K \).

Since \( N \geq 2^K \), it will repeat at least \( K \) times for \( N \). But it cannot repeat \( K + 1 \) times: \( 2^{K+1} \) is the smallest number for which this loop repeats \( K + 1 \) times, and \( N \) is smaller than \( 2^{K+1} \). Therefore, if \( N \) is bounded by the \( K \)th and \((K+1)\)st powers of two, the loop will repeat \( K \) times.

What is the relation between \( K \) and \( N \)? Taking \( \log_2 \) on equation 5.1 gives:

\[
K \leq \log_2 N < K + 1
\]  

(5.2)

That is, \( K \) is the integer part (floor) of the \( \log_2 \) of \( N \). Thus, the number of times a number \( N \) can be divided by 2 before it will reach 1 is the integer part \( \log_2 N \).

There is a distinct difference however between this proof and the behaviour of the error tracking algorithm. As described earlier in this section, since all intervals need to be of integer length\(^4\), the number of iterations (\( K \)) will differ depending on the location of the error. The above proof however is directly applicable in situations where \( n_0 = 2^i \), \( 0 \leq i \in \mathbb{N} \), and will later help us to determine worst-case scenarios of the error-tracking behaviour.

**Analysing the Error Tracking Behaviour: No Errors**

Let us introduce two variables:

\(^4\)That is, an odd number will be split in two intervals of different length.
The cost $\alpha$ of reading a single record from the database. This is without considering any overhead costs from establishing connections, making method calls, etc.

The cost $\beta$ of making a new database call, possibly including establishing a connection. These costs $\alpha$ and $\beta$ can represent network load, processing needs, a combination of the two, or something completely different. What is interesting is the relation between the two.

When in the following sections analysing InfoBroker’s error tracking algorithm, two quantities will be derived:

- The total cost of reading records from the database, $c_\alpha$.
- The total overhead cost of making calls to the database, $c_\beta$.

In the case of no errors, $e = 0$, deriving $c_\alpha$ and $c_\beta$ is trivial:

\[
c_\alpha = \alpha n_0 \quad \forall n_0 \in \mathbb{N} \tag{5.3}
\]

and

\[
c_\beta = \beta \tag{5.4}
\]

That is, one single iteration will result in $n_0$ records being read from the database.

Given that a certain number of records $y$ are to be read from the database, the minimum total cost $(c_\alpha + c_\beta)$ in case of no errors is always reached when $n_0 = y$. Coming to this conclusion is trivial: $c_\alpha$ is linear to $n_0$ and splitting the interval $y$ in two database calls, $n_0 = \frac{y}{2}$, leads to the same total cost $c_\alpha$:

\[
c_\alpha = ay = a \frac{y}{2} + a \frac{y}{2} \tag{5.5}
\]

However $c_\beta$ adds a constant factor $c_\beta$ for each database call, thus splitting the interval $y$ in two will result in double $c_\beta$.

Reality however is never plain and simple. Bluntly choosing $n_0 = y$ can be unwise for several reasons:

- Every computer system has limitations in physical memory and processing power. Trying to handle too much data at a time may have effects on system performance.
- Errors can occur, also when not expected.

So what if errors do occur? How will $c_\alpha$ and $c_\beta$ be affected? This will be discussed in the following sections.

**Analysing the Error Tracking Behaviour: Single Error**

Again, refer to Figure 5.2 for an example of the error-tracking algorithm finding a single error, in this case when $n_0 = 8$. Reordering the iterations in a no-longer chronological way, grouping them by the size $n$ of the parts yields Figure 5.3. The first row represents the first iteration, a single read of 8 records. The next rows each represent two reads of 4, 2 and 1 records respectively. In total, how many calls are made to the database? Recall that finding the error involves repeatedly splitting the interval $n$ in two until $n = 1$ and errors still occur. Two database calls will be made at each iteration, one for the iteration that finishes successfully, and one for the iteration that fails. How many iterations does it take to reach $n = 1$? Given $n_0$ is a power of two, the answer is $\log_2 n_0$, as was previously derived from Equation 5.1.

Supported by Figure 5.3, the total number of database calls is thus $1 + 2 \log_2 n_0$, including the first iteration spanning the entire original interval $n_0$. 

57
Thus, the total cost $c_{\beta}$ of database calls is:

$$c_{\beta} = \beta \left(1 + 2 \log_2 n_0\right) \quad \forall n_0 = 2^j, 0 \leq j \in \mathbb{N} \quad (5.6)$$

How many records are read in total? Refer to the right-hand comments in Figure 5.3. The total number of records read are:

$$2 \cdot 1 + 2 \cdot 2 + 2 \cdot 4 + 1 \cdot 8 = 22 \quad (5.7)$$

The pattern is clear: two reads of $n = 2^0$, $n = 2^1$ and $n = 2^2$ respectively and a single read of $n = n_0$. This pattern is general. For example in the case of $n_0 = 16$, the total number of records read will be $2 \cdot 1 + 2 \cdot 2 + 2 \cdot 4 + 2 \cdot 8 + 1 \cdot 16$. In general form, the total number of records read at a single error in $n_0$ records is:

$$\eta = n_0 + 2 \sum_{i=0}^{\log_2 n_0 - 1} 2^i \quad \forall n_0 = 2^j, 0 \leq j \in \mathbb{N} \quad (5.8)$$

This is a geometric series. Applying the well-known [1] equation

$$\sum_{k=0}^{n-1} ar^k = \frac{a (1 - r^n)}{1 - r} \quad (5.9)$$

to Equation 5.8 yields

$$\eta = n_0 + 2 \left(1 - 2^{\log_2 n_0}\right) \quad \forall n_0 = 2^j, 0 \leq j \in \mathbb{N} \quad (5.10)$$

and after simplification:

$$\eta = 3n_0 - 2 \quad \forall n_0 = 2^j, 0 \leq j \in \mathbb{N} \quad (5.11)$$

The total cost $c_{\alpha}$ equals $\alpha \cdot \eta$, that is:

$$c_{\alpha} = 3\alpha \left(n_0 - \frac{2}{3}\right) \quad \forall n_0 = 2^j, 0 \leq j \in \mathbb{N} \quad (5.12)$$

Figures 5.4 and 5.5 plot $c_{\alpha}$ and $c_{\beta}$ respectively for $1 \leq n_0 \leq 512$, $\alpha = \beta = 1$, calculated using Equations 5.12 and 5.6. The circles represent calculated values. Between the circles, $c_{\alpha}$ and $c_{\beta}$ will take values depending on the physical location of the error. Values will still be bounded however by the value of the next circle to the left and right respectively, as discussed in Section 5.2.2.

In the next section, the error tracking behaviour in a worst-case scenario will be discussed. After that, a comparison of the different scenarios will be presented and some conclusions will be drawn, and finally, some improvements to the error-tracking algorithm will be suggested.
Figure 5.4 The total number of records read for $n_0 = 2^i, i = 0 \ldots 9$, single error. If $\alpha = 1$, this is also equal to the cost $c_\alpha$. 
Figure 5.5 The total number of database calls for $n_0 = 2^i$, $i = 0 \ldots 9$, single error. If $\beta = 1$, this is also equal to the cost $c_0$. 
Analysing the Error Tracking Behaviour: Maximum Number of Errors

Imagine a situation where every single record will lead to at least one error, due to for example a misconfiguration in the **ris**. How will this affect InfoBroker’s behaviour?

Assume \( n_0 = 8 \). Figure 5.6 shows the iterations in chronological order, Figure 5.7 grouped by \( n \). Let us start by deriving \( c_\alpha \). As can be seen in Figure 5.7, \( n_0 \) records will be read as many times as \( n_0 \) can be divided by two, which is \( \log_2 n_0 \), plus the first iteration. Thus, written in general form the equation is:

\[
c_\alpha = an(\log_2 n_0 + 1) \quad \forall n_0 = 2^i, 0 \leq i \in \mathbb{N}
\] (5.13)

To help deriving \( c_\beta \), we refer to the right-hand comments in Figure 5.7. The pattern is clear: 8 reads of 1 record respectively, 4 reads of twice that, 2 reads of 4 records, and a single read of the full interval \( n_0 \). As long as \( n_0 \) is a power of two, this is a general pattern.

In this case, \( 1 + 2 + 4 + 8 \) reads. The general form is:

\[
c_\beta = \beta \sum_{i=0}^{\log_2 n_0} 2^i \quad \forall n_0 = 2^i, 0 \leq i \in \mathbb{N}
\] (5.14)

Combining this expression with Equation 5.9 yields

\[
c_\beta = \beta \left( 1 - \frac{2^{\log_2 n_0 + 1}}{1 - 2} \right)
\] (5.15)

and after simplification:

\[
c_\beta = 2\beta \left( n_0 - \frac{1}{2} \right)
\] (5.16)

Figures 5.8 and 5.9 plot \( c_\alpha \) and \( c_\beta \) for \( 1 \leq n_0 \leq 512, \alpha = \beta = 1 \), calculated using Equations 5.13 and 5.16. The circles represent calculated values.

Comparing the Scenarios: Choosing \( n_0 \)

In this section, a comparison of worst-case-scenarios for the three cases \( e = 0, e = 1, \) and \( e = n_0 \) will be presented. How should \( n_0 \) be chosen with knowledge of the equations derived?

Figures 5.10 and 5.11 plot total number of records read and total number of database calls for the three cases \( e = 0, e = 1, \) and \( e = n_0 \). As expected, the worst thinkable case, \( e = n_0 \), generates significantly more database calls and records to be read, and thus notably higher costs \( c_\beta \) and \( c_\alpha \). In reality however, this will not happen unless something is seriously wrong.

Where do the errors come from? In a fully functional and bug-free system: from human mistakes! In a perfect world, the checks Feedback performs in ensuring the data is valid can and will be done in the **ris** user-interface when the doctor or the medical secretary enters the data. For example, it should not be possible to set the date of a performed diagnosis to a future date. However since InfoBroker collaborates with a number of **ris** from a number of different vendors, no assumptions of this kind can be made.

How often do the errors occur? This will be dependent on a number of factors, including characteristics of the **ris** and the general working procedures at the clinic.

Choosing \( n_0 \) will be dependent on the error frequency but also on the relation between \( \alpha \) and \( \beta \).

In the case \( \alpha \ll \beta \), that is, the cost of making a database call is much larger that the cost of reading a record, one would choose \( n_0 \) as large as the system can handle if none or few errors are expected. Recall that in the case of \( e = 0 \), \( c_\beta \) is constant for all \( n_0 \) while \( c_\alpha \) grows linear to \( n_0 \). On the other hand, the combination of a high cost \( \beta \) and a large number of
Figure 5.6 An example of InfoBroker’s error tracking algorithm resolving errors at every single record ($n_0 = 8, e = n_0$).

Figure 5.7 The database reads when recursively finding a single error in $n_0 = 8$ records, grouped by the size $n$ of the respective chunk being read.
Figure 5.8 The total number of records read for $n_0 = 2^i, i = 0 \ldots 9$, maximum number of errors. If $\alpha = 1$, this is also equal to the cost $c_\alpha$. 

\[ \frac{\text{Total number of records read, maximum number of errors}}{n_0} \]
Figure 5.9 The total number of database calls for $n_0 = 2^i$, $i = 0 \ldots 9$, maximum number of errors. If $\beta = 1$, this is also equal to the cost $c_\beta$. 
Figure 5.10 The total number of records read for $n_0 = 2^i, i = 0 \ldots 9$, in the three cases of $e = 0$, $e = 1$ and $e = n_0$. 
Number of database calls: no error, single error and maximum number of errors

- **Single error (e=1)**
- **Maximum number of errors (e=n_0)**
- **No errors (e=0)**

*Figure 5.11* The total number of database calls for \( n_0 = 2^i, i = 0 \ldots 9 \), in the three cases of \( e = 0, e = 1 \) and \( e = n_0 \).
errors \((e > 1)\) is potentially very harmful to system performance, as implied in Figure 5.11. In a real-life scenario, where \(\alpha\) and \(\beta\) reflects network usage and perhaps more importantly, processing needs at Feedback and ris, it must be assumed that \(\beta\) is significantly larger than \(\alpha\), not unlikely by a factor of 50 or even 100.

The suggested method for choosing \(n_0\) is:

1. On a new system setup, where \(e\) is unknown, start with a low \(n_0\), preferably \(n_0 \leq 128\).
2. As soon as an error frequency can be determined (from InfoBroker’s logfiles), increase \(n_0\) to a suitable value based on the relation between \(\alpha\) and \(\beta\), the error frequency and the equations and figures derived in this chapter!

**Obvious Improvements to the Error-Tracking Algorithm**

Two obvious improvements can be made to the InfoBroker error-tracking algorithm described in this chapter:

1. In the cases of rejected data errors, that is, Feedback does not accept the data, there is little point in always re-reading the records from the database between iterations. Rather, records that have been read from the database can be cached in physical memory between iterations. In theory, this would prevent records from being read more than once and \(c_\alpha\) as well as \(c_\beta\) can approach those corresponding to \(e = 0\).

   In reality however rejected data errors may originate from not only invalidities with the physical data but also database errors and misconfigurations. Caching data known to contain invalid elements needs to be done with great caution and is beyond the scope of this thesis.

2. In many cases, Feedback as well as the database will be able to report back not only an error but also a more detailed description of where the error originates from. Is so, InfoBroker can use that information to find and mark the errors. A drastic alternative is to blindly trust the error messages — mark the affected records as bad or rejected respectively and then carry on with other records. A less dramatic alternative is to use the supplied information as a hint when splitting an interval. Splitting \(n\) in half is a good alternative if nothing is known about the error, however splitting the interval close to where the error is can help minimizing the number of iterations needed. And if the guessing is wrong, the error will still be found, however at the possible cost of one extra iteration.

   Such an improvement is highly suggested but will not be further discussed in this thesis.

**5.2.3 InfoBroker Scalability**

The Java platform is designed with scalability in mind. Multi-threading, for example, is tightly integrated in Java and properly designed software will scale well on large, multi-processor server clusters.

The current InfoBroker implementation is threaded largely by the underlying rmi framework. A new thread is spawned for each client request, and all class implementations are thread-safe to handle requests from multiple concurrent threads.

All i/o is asynchronous where feasible. Locking and synchronization of methods\(^5\) have been kept at a minimum for maximum concurrency and minimum overhead (making a synchronized method call is roughly 20 times more expensive than a non-synchronized call [18]).

---

\(^5\)Declaring a method synchronized in Java makes it atomic to one thread, that is, only thread at a time is allowed to execute the method. This is used to protect shared resources.
5.2.4 InfoBroker Security

The interpreted nature of the Java platform comes with both advantages and disadvantages. The main drawback is performance, since bytecode has to be run through an interpreter rather than being executed natively. On the positive side, the interpreter keeps Java programs from overrunning the runtime stack, commonly leading to security problems, and enables fine-grain control of what Java applications are allowed and not allowed to do. Unless a permission is explicitly granted, the code cannot access the resource that is guarded by that permission.

The security-related bugs found so far in the Java platform have been very subtle and relatively few in nature [20] and Java must be considered one of the most, if not the most secure programming languages to date.

At the point of writing this thesis, one agent has been implemented, tailored to the ris product Rados, maintained by Swedish vendor WM-data. The level of security in InfoBroker-Rados communication is determined by Rados rather than InfoBroker as Rados defines the interface to access data (based on sql calls over Oracle sq*net). To overcome the shortcomings, that is, lack of a encryption and proper authentication schemes, InfoBroker must be deployed close to Rados on the computer network, preferably on the same segment, ideally using a private network shared only between the computer running InfoBroker and the computer running Rados.

The InfoBroker-Feedback communication is based on Java RMI over TLS sockets. This is implemented using Java custom socket types, tailoring a custom “socket factory” that creates sockets of the desired type, rather than using the default socket factory that creates standard tcp sockets. The TLS implementation used in InfoBroker is based on the standard extension Java Secure Sockets Extension [72], jsse. jsse implements full TLS and ssl version 3 standards, including x.509 certificates for authentication. However rather than using a trusted certificate authority (CA) such as VeriSign or Thawte, certificates are signed with a secret-key kept by eCare. This enables fine-grain control over what certificates are issued and allowed between Feedback and InfoBroker. The public-key of the CA (which is eCare) of course needs to be distributed and is thus bundled with both Feedback and InfoBroker.

5.2.5 InfoBroker Event and Error Logging

<table>
<thead>
<tr>
<th>Global events</th>
<th>System-wide or otherwise general events.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorMailer events</td>
<td>Events originating from an ErrorMailer thread.</td>
</tr>
<tr>
<td>LogMailer events</td>
<td>Events originating from a LogMailer thread.</td>
</tr>
<tr>
<td>RIS agent events</td>
<td>Events originating from a ris agent. This is a rough classification. Most ris agent implementations will define a number of sub-events.</td>
</tr>
<tr>
<td>PACS agent events</td>
<td>Events originating from a PACS agent. Similarly to ris agents, PACS agents will in general define a number of agent-specific sub-events.</td>
</tr>
</tbody>
</table>

Table 5.2 InfoBroker event types.

Any server-oriented application, in particular in a distributed environment, should provide the option of logging system events. To section proposes a framework to enable flexible logging in InfoBroker.

6http://www.wmdata.se
InfoBroker logging is based on the assumption that events generally can be classified using a two-dimensional matrix as in Figure 5.12. The event type is one of a number of InfoBroker specific pre-defined events, listed in Table 5.2. The importance is a number 0 . . . 127 where 127 is maximum importance.

The developer is responsible for positioning her events in Figure 5.12. That is, every event is assigned to an event type and a given a number corresponding to its “importance”.

The point of all these efforts is to give the system administrator control over what is logged, and what is not. The logging is configured using a xml-based ruleset with explicit and implicit rules, for example a general rule “log everything with importance >63” combined with two specific rules that overrides the general rule: “do not log ErrorMailer events” and “log all rus agent Rados events”. For an example visualisation of the three example rules, using the two-dimensional matrix, see Figure 5.13. The shaded area covers events that will be logged.

An intuitive gui for the system administrator can be designed using this model for visualizing rulesets. This way the administrator can, by drawing simple boxes on a workspace, decide what will be logged and not. The graphical client will generate the rulesets to be used
by the system. Alternatively, a more textfile-oriented system administrator may prefer to write the rulesets from scratch.

Yet another thing needs to be configured: where do the logged events go? Typical examples would be fed into a database, written in plain text files, printed on a screen, or a combination of them all. The implementation needs to be flexible and support logging through zero or many channels, configured at runtime.

Current InfoBroker implementation support only a small portion of what has been discussed in this section. The events are classified according to type but not importance, and currently all events are logged. No XML-syntax has been specified to define rulesets and hence no parser has been implemented. The latter is trivial however using standard XML support in Java.

The actual logging of events are implemented using adapters that can easily be plugged into the logging framework. A stream-writer adapter has been implemented, capable of piping events into a Java OutputStream, thus capable of logging to files, on the screen, over a network, or any other feasible output channel supported by Java. A default configured InfoBroker will output its events on screen and in logfiles that will periodically be compressed and e-mailed to the system administrator by a LogMailer thread.
Chapter 6

Final Comments and Conclusions

This thesis has suggested CORBA as long-term candidate for Feedback directory services, as well as the data propagation engine between Feedback servers.

At the moment, commercial CORBA implementations for Java are often parts of applications servers also providing full J2EE support. The latter, including Enterprise JavaBeans (EJB), have rightfully received a great deal of attention in the IT industry and the enthusiasm for CORBA has lately been somewhat eclipsed by J2EE.

That said, it is important to remember that large parts of J2EE are based on CORBA services. It is unlikely however that Sun will ever offer pure CORBA services; rather, Sun is pushing for greater integration between EJB and CORBA. Not surprisingly, the CORBA 3.0 specification includes features designed to further enhance CORBA-Java interoperability.

Meanwhile, Java RMI is progressing towards being CORBA compliant with the possibility to run RMI over CORBA's protocol IIOP.

For eCare, and the rest of the IT industry, this is fortunate. Very likely, eCare will be able to use the powers of J2EE and EJB without having to sacrifice language-neutrality. As long as CORBA is the connecting technology, components written in other languages can easily be plugged into the application.

So, referring back to the original question: yes, it is possible to build a self-configuring directory service under the constraints that were sketched in Chapter 1 and specified in Chapter 3. However not using an off-the-shelf product but rather using state-of-the-art development tools for distributed applications such as CORBA, possibly in combination with EJB.

Feedback servers can share data in a flexible and loosely coupled manner using a combination of XML and Java RMI for short-term solution, and a combination of CORBA and XML for a long-term solution. This thesis has also shown that it is possible to implement connectivity with several different legacy RIS and PACS. The solution is named InfoBroker and is already capable of integrating with the RIS product Rados.
# Appendix A

## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES</td>
<td>Tripple Data Encryption Standard</td>
</tr>
<tr>
<td>ADO+</td>
<td>ActiveX Data Objects “plus”</td>
</tr>
<tr>
<td>ASP+</td>
<td>Active Server Pages “plus”</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DAP</td>
<td>Directory Access Protocol</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DIT</td>
<td>Directory Information Tree</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
</tr>
<tr>
<td>DSML</td>
<td>Directory Services Markup Language</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
</tr>
<tr>
<td>FDS</td>
<td>Feedback Directory Service</td>
</tr>
<tr>
<td>FDSP</td>
<td>Feedback Directory Service Protocol</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol over ssl</td>
</tr>
<tr>
<td>IDEA</td>
<td>International Data Encryption Algorithm</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>IOP</td>
<td>Internet Inter-orb Protocol</td>
</tr>
<tr>
<td>IOPS</td>
<td>Internet Inter-orb Protocol Secure</td>
</tr>
<tr>
<td>IL</td>
<td>Internal Language (Microsoft .NET)</td>
</tr>
<tr>
<td>IOR</td>
<td>Interoperable Object References</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPSEC</td>
<td>Internet Protocol Security Protocol</td>
</tr>
<tr>
<td>IPX</td>
<td>Internetwork Packet Exchange</td>
</tr>
<tr>
<td>IPX/SPX</td>
<td>Internetwork Packet Exchange / Sequenced Packet Exchange</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform, Enterprise Edition</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>JSSE</td>
<td>Java Secure Sockets Extension</td>
</tr>
<tr>
<td>JTS</td>
<td>Java Transaction Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LDIF</td>
<td>The LDAP Data Interchange format</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-Oriented Middleware</td>
</tr>
<tr>
<td>NetBEUI</td>
<td>NetBIOS Extended User Interface</td>
</tr>
<tr>
<td>NetBIOS</td>
<td>Network Basic Input/Output System</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PACS</td>
<td>Picture Archive and Communication System</td>
</tr>
<tr>
<td>RIS</td>
<td>Radiology Information System</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest, Shamir, Adleman</td>
</tr>
<tr>
<td>SMB</td>
<td>Server Message Block</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol over Internet Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>Extensible Markup Language – Remote Procedure Call</td>
</tr>
</tbody>
</table>
Bibliography


BIBLIOGRAPHY

Index

ADO+, 15
API, 47, 48, 50
ASP+, 15
c#, 15
cics, 11
corba, i, 9, 11–15, 19, 20, 23, 24, 29, 30, 37, 40–43, 48–51, 54, 71
dap, 21
dce, 11
des, 17, 48
dit, 21
dns, 20–22, 28–31, 33, 35, 37
don, 21
dsmi, 22
ejb, 14, 71
fds, iii, 30–42
ftp, 24, 47, 48, 50
gui, 52, 69
html, 3, 14, 47, 51
http, 11, 15, 18, 24, 45–50
ibm secureway directory, 22
idea, 17
idl, 10, 12, 41, 42
iiop, 12, 13, 23, 30, 41, 42, 48–50, 71
il, 15
ior, 12, 13
ipx/spx, 12, 22, 29
ipx, 22
ip, 4, 12, 18, 19, 23, 35, 37, 39
ipsec, 18, 19
jaxp, 47
jms, 16, 17, 46, 47
jsp, 14
jsse, 68
jts, 14
ldap, 21, 22, 28, 30, 33, 35
ldif, 22
mom, 15, 16, 18, 19, 24, 46–51
mqseries, 16, 46
netbeui, 22
netbios, 22, 29, 30
omg, 12–14, 41
orb, 11–13, 18, 23, 41–43, 48, 50
osi, 21
pacs, 2, 3, 25, 26, 45, 51, 52, 54, 68, 71
pdf, 47, 51
ris, 2, 3, 25, 26, 45, 51, 52, 54, 55, 61, 67–69, 71
rmi, 3, 11, 19, 42, 43, 45, 47, 49–51, 54, 67, 68, 71
rmi transport protocol, 49
rpc, 10–12, 19, 23
rsa, 17, 18
sjunet, 4
smb, 22, 29, 30
soap, 15
socks, 13
sql, 9, 10, 19
ssh, 24, 47, 48, 50
ssl, 18, 19, 28, 30, 48–50, 68
tcp/ip, i, 4, 12, 21, 22
tcp, 18, 23, 42, 45, 46, 48, 68
tir/rendezvous, 16
tls, 18, 19, 48–50, 68
udp, 18
unix, 7, 10, 22, 52
url, 21
wins, 22, 29
xml-rpc, 11
xml, 11, 15, 22, 46–51, 69–71
rmi transport protocol, 45
3des, 17, 48
asymmetric key cipher, 17
asynchronous communication, 10
atomic transactions, 8
ciphers, 17
coupling, 10
directory service, 20
distributed transaction monitor, 9
INDEX

domain weight, 38
Feedback Tree, iii, 27–37, 40–43
InfoBroker, i, iii, 45, 51–57, 61, 62, 67–71
iPlanet Directory Server, 22
Java Messaging Service, 16
Java Object Serialization, 45
Java Secure Sockets Extension, 68
Microsoft .NET, 9, 15, 19
naming service, 20
OpenLDAP, 22
Oracle Advanced Queueing, 16
Oracle Internet Directory, 22
processing dependencies, 19
Rados, 68
Samba, 22
symmetric key cipher, 17
synchronous communication, 10
Tuxedo, 11
virtual private network, 18
X.509, 17, 18, 38, 68