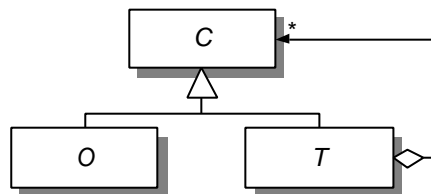


*Experiences from extending a legacy
system with CORBA components*
COT/4-08-V1.3



Centre for Object Technology

*Centre for
Object Technology*

Revision history: V1.0 1999-03-01 First draft.
 V1.1 1999-05-18 Second draft to review.
 V1.2 1999-06-21 Third draft to review.
 V1.3 1999-10-01 Final version.

Author(s): Allan R. Lassen, RAMBØLL
 Jacob Steen Due, RAMBØLL
 Emil Hahn Pedersen, RAMBØLL
 Mohammad Al-Shamri, RAMBØLL

Status: Final

Publication: Public

Summary:

A presentation of the experiences gained by performing an extension of a legacy system by using object-oriented technology. The experiences are the basis of a general approach to extend a legacy system with add-on functionality based on the CORBA technology.

© Copyright 1999

The Centre for Object Technology (COT) is a three year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Danish National Centre for IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:
Maersk Line, Maersk Training Centre, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Steel Shipyard, A.P. Møller, University of Aarhus, Odense University, University of Copenhagen, Danish Technological Institute and Danish Maritime Institute

1. Introduction

In the IT industry many systems, developed a long time ago, are still maintained because they are crucial for the IT business. Introducing new and better technology in the development process raises a challenge on how to adopt the new technology in the older systems. In this document we present our experiences of extending a system using object-oriented technology and CORBA. For more information on our project please refer to [3] which contains the design of the new system.

2. Goal of project

The goals of this project were first of all to gain experience and secondly to produce a final product. The main factors were:

- We saw a need to develop a method that allows new functionality to be added to older legacy¹ systems.
 - The new functionality should be in the form of add-on modules.
 - The changes needed in the legacy system should be minimised (and ideally there should be no changes at all in the legacy system).
- The architecture should allow implementation in multi-tier architectures.
 - We wanted to implement it in CORBA using the latest version of Oracle Application Server (OAS). This was basically a commercial reason. We had good experience with other Oracle tools and we expected that OAS would integrate nicely with those Oracle databases that many of our systems use.

The project is part of the research conducted in the Centre for Object Technology [1]. The actual project is part of industrial case 4 concerning integration with non-OO systems. By using object-oriented technology a partial goal of the project was to examine if the above-mentioned requirements could be realised successfully.

The basis of the project was a workflow and document management system, WorkSAFE, developed by RAMBØLL [2] as a standard system. Although the system is relatively new and well documented it corresponds to the definition of legacy system used in this report.

The current workflow system is designed to be used internally in an organisation. We wanted to give external partners access to selected parts of the system (i.e. access to some of the documents and to participate in the workflow for quality assurance). Partners should learn the system quickly. Eventhough the current system has a very general user interface, it takes time to learn and we wanted to supply a user interface designed to meet the requirements of the partners.

¹ In this report we use the term 'Legacy system' to denote monolithic systems. Typical examples are custom developed systems for various purposes such as accounting and other administrative purposes. Such systems are traditionally difficult and expensive to develop and maintain.

Whereas this may seem as a rather trivial case it does in fact raise a number of interesting questions that have turned out to produce implications in general when adding new functionality to legacy systems:

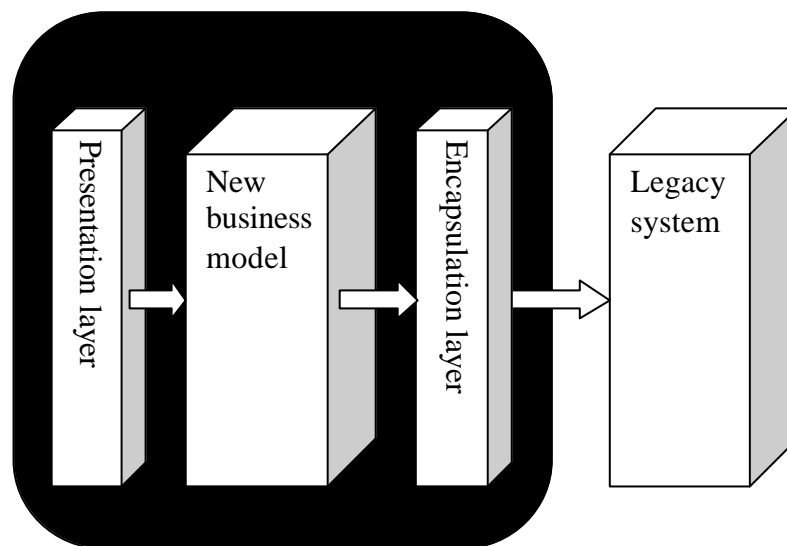
- How do we get a maintainable interface to the legacy system?
- How do we reduce our dependency on changes in the legacy system?
- How does the choice of technology influence our implementation?

3. Architecture

A key issue in adding new functionality to the legacy system, is the architecture to be used for the add-on modules. You may ask yourself the question: What kinds of interfaces are available for the legacy system?

Many legacy systems are using a relational database as persistent storage. The database tables implement an entity-relationship model, and usually you can easily transform it into an object model by representing each entity as a class. However, in our case the system had a complicated E/R-model because it used a meta-model of all its data. Therefore we decided to use an interface based on its Oracle database views and stored procedures. In another system you may have other alternatives, such as a documented API.

In our project we came up with a logical architecture, which we in general find useful. The architecture is illustrated in the figure below.



The encapsulation layer contains the classes corresponding directly to the legacy system interface. The layer is (and must be) as simple as possible and contains only the functionality that exists in the legacy system. The purpose of this layer is to shield the new business model from the complexities and idiosyncrasies in the actual implementation of the legacy system. Therefore it is very important to make the layer as simple as possible.

The new business model is a model of the relevant parts of the legacy system and contains classes with business logic of the new functionality. The classes use the encapsulation layer to perform actions towards the legacy system. The business classes serve as an abstraction of the legacy system for the user interface and they only contain information that is relevant to the new clients.

In a traditional multi-tier architecture there is a database tier and one or more business application tiers. In our architecture the business logic is performed in a combination of the new business model and the legacy system.

The presentation layer implements the user interface like in traditional multi-tier architectures.

3.1 Considerations about the clients

In our project the new functionality did not depend on the existing user interface and we considered three solutions to the presentation layer:

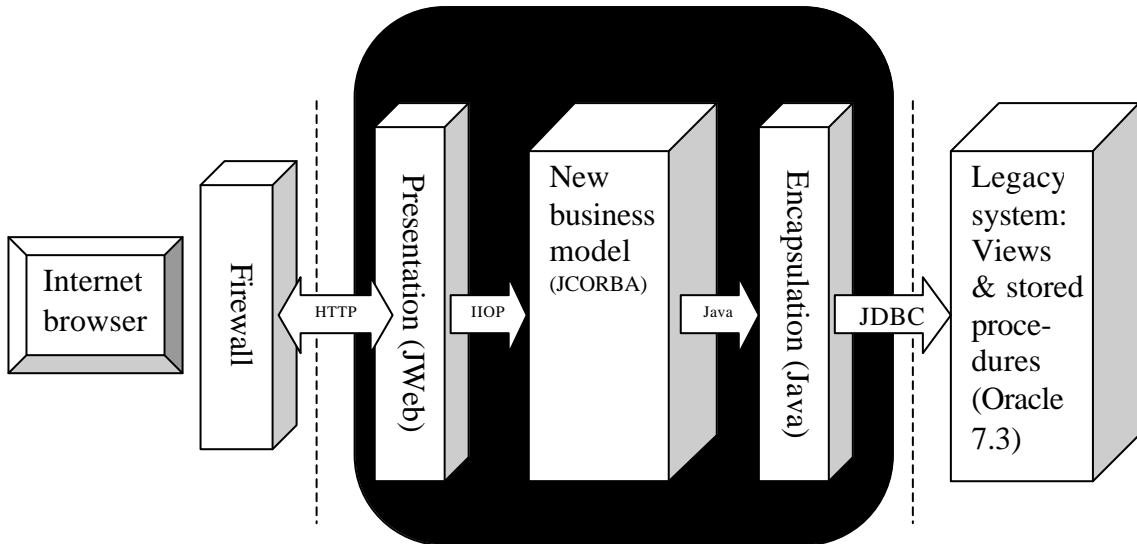
- An HTML based presentation where HTML pages are generated from the presentation layer on a server and displayed in an Internet browser on the client.
- A presentation using Java applets running within the Internet browser. The applets are Java classes that will be loaded from a server by the browser and then run on the client. The applets will communicate through CORBA IIOP with business classes running on a server.
- A stand-alone client application e.g. developed in Java. The application does not require a browser and will communicate with business classes in the same way as Java applets. Some mechanism to install the application would be needed.

We believe the Java applets or application would offer the best facilities for implementing a nice and fancy user interface. However to have our end-users, i.e. a partner, accessing the system through the Internet raises a problem with firewalls. Currently most firewalls do not allow the CORBA IIOP protocol to pass through. Eventhough, we get our company firewall to work with IIOP the user will probably sit behind a partner firewall that we not are able to control. Therefore we decided to dynamically generate HTML from our presentation layer.

An alternative is to implement a Java based user interface by marshalling and tunnelling the Java objects through the HTTP protocol. But it requires some extra work to implement and we didn't want to struggle with this issue in our project. Hopefully the problem about getting the IIOP protocol through firewalls will soon be solved and thus make implementation of fancy user interfaces easier.

3.2 Used technology

The Oracle Application Server 4.0 is the platform of our implementation of the architecture and we use techniques from the product as illustrated below:



The figure shows a physical model of our new system. All the classes in the system are implemented in Java. The business classes are subclasses of a JCORBA superclass, which is a class that Oracle has derived from the standard OMG CORBA class. JWeb classes are classes that dynamically generate HTML pages from Java. The dotted, vertical lines separate the parts that can be placed on separate physical servers.

4. Development process

4.1 Modelling the problem domain

The architecture described in the previous section naturally leads to the following steps in the modelling.

- An external model of the legacy system must be created in order to provide the encapsulation layer.
- A new business model is made.
- The encapsulation and business models are mapped together.

In our case it was relatively easy to construct the encapsulation layer because of the interfaces available to the legacy system, but this is not always the case. In other systems the nature and structure of the legacy system may make the task more complicated, e.g. if the only way to enter and query information in the legacy system is through its user interface. Though the goal is to use existing code from the legacy system there is a risk on having to establish some of the functionality in the encapsulation layer.

The business model could be created with only little awareness of the old legacy system and with focus on the application field. In effect this model was created in the same way as creating a model for a completely new system. Of course the model has a constraint that it must conform to the legacy model.

Interestingly enough, we found out that eventhough the developers with real knowledge of the legacy system found it relatively easy to create the encapsulation layer, they also found it demanding to create the business model due to the difficulty of abstracting from the familiar terms used in the legacy system. Actually, the participants with superficial knowledge of the legacy system were the ones who made the greatest progress on the business model. In practice two tracks were formed to develop the layers in parallel and this worked surprisingly well because of the clear division of the responsibilities in the layers.

In our case the business model did not introduce the need for extra information - it could all be stored in the legacy system. Therefore the problems associated to such a need have not been an issue. If such a need occurs the correct place for storing the information is in the business model when it is required that the legacy system shall not be modified. However, the complexities in securing transactional control, data security and performance are not described in this project, eventually an exercise for the reader ☺.

The objects in the business model may be distributed when they are implemented as CORBA objects. This raises an interesting opportunity to obtain better fault tolerance in the system. However we didn't investigate this topic further in our project.

The mapping between the encapsulation and business models proved to be relatively easy due to several reasons. One reason is the major factor that the problem domain for the new functionality was close to the problem domain of the old legacy system. This must be assumed to be the case in most situations where add-on functionality is put on top of legacy systems. Honestly, our legacy system was developed with OO techniques, as an implementation of an OO model in the relational paradigm. Though this implementation probably reduced the complexity in mapping the models we still believe our experience is correct because we interface to the legacy system as a traditional system implemented in a relational database.

4.2 Functionality

In developing the object structure of the business model and methods we had good experiences with small descriptions of scenarios². The scenarios illustrated the work processes we wanted to support and they were combined with the involved dialog flows. The scenarios and dialog flows helped to identify key methods and events in the system.

² The scenarios descriptions could have been replaced by the standardised use cases from UML but we chose to use descriptions that were less formal.

4.3 Design

After creating a good business model we found it difficult to describe a detailed design without real knowledge on the implementation platform. It is a general challenge when you take new technology in use. We believe it is a good idea to try some experiments and perhaps find some examples to benefit from. Our project described in [3] may serve as an example. It is also a good idea to see if there is a design pattern describing a solution to similar problems. An introduction to patterns is given in [4].

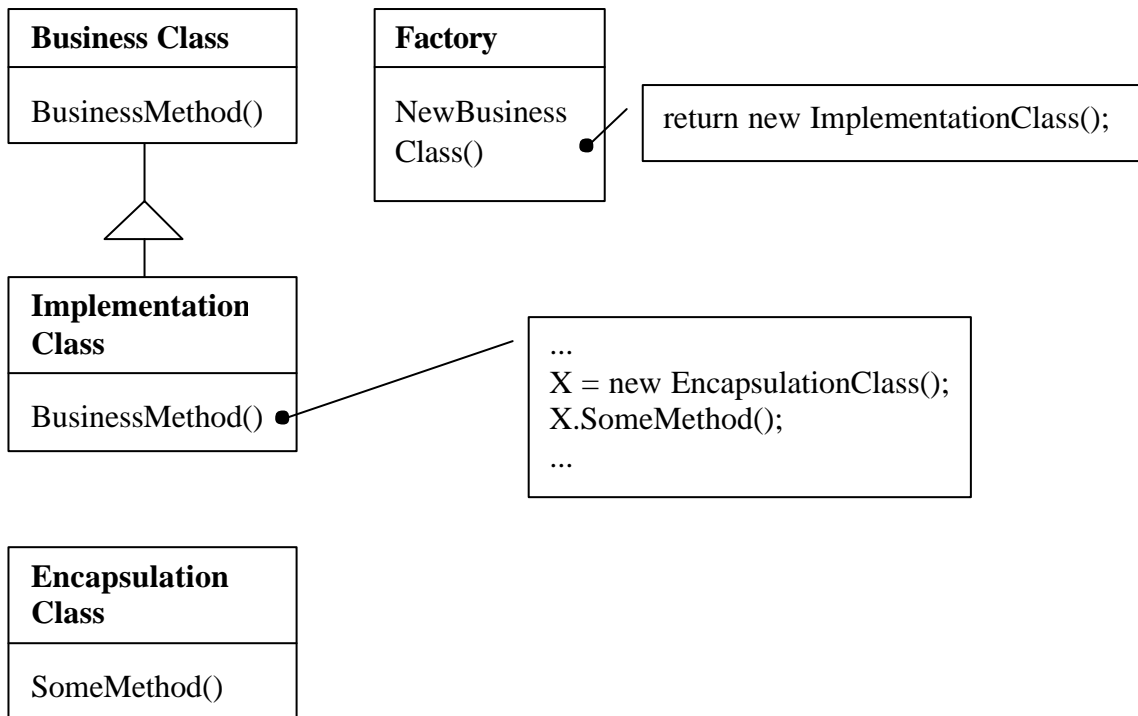
One of our concerns was how the clients should connect to the system and how we could keep track of the connection between user interactions. We found a good solution by introducing a session class and a session manager that handles the user connections (see [3] for details).

Our object-oriented system uses the relational database in the legacy system as persistent storage. It raises a question on how to manage the information when it is represented both in the relational database itself and in the object instances that we create in our system. We wanted to be certain that the information was updated correctly in our system as well as in the legacy system. Indeed, it is a problem if the legacy system changes the object values after our system have loaded them into an object instance or vice versa. We were lucky that the legacy system had implemented a locking strategy we could use to notify when we were going to update the information. In another system you may have to struggle with this issue.

In order to handle the information in the legacy system we used the following two general principles:

- We introduced an object manager class for each class that would be manipulated in our new system. The manager implemented methods to create, retrieve and store objects while ensuring the integrity between the two systems. The manager methods also checked and ensured that a particular object was only retrieved once. A manager class must only be instantiated once and we ensured this by using the Singleton pattern [5].
- Classes that could not be updated in our system were implemented as data banks. The data banks provided methods for looking up information in the legacy system. Persons and companies were not administrated by our system and we thus implemented a data bank for persons and a data bank for companies.

The mapping between business classes and encapsulation classes leads to high coupling between our new functionality and the legacy system. We solved this problem by using a Factory Method pattern [5]. Our business classes describe the general interface used by the presentation layer and they only implement general methods. Methods that depend on encapsulation classes are implemented in derived implementation classes. The principle is illustrated in the figure below:



This technique proved to be very useful. In principle, we are able to transform our system to use another equivalent legacy system.

5. Implementation

During the implementation, we experienced that it was comparatively easy to write the Java/CORBA code used to implement our design in spite of the fact that our knowledge of these technologies was limited.

Usually you do not have to think of a Java object being a CORBA object or not. But when parsing objects between CORBA objects you must be aware that only CORBA defined types (either standard types or IDL defined types) can be used as parameters to CORBA objects. A plain native Java object cannot be used. A JDBC database connection is a native Java defined class, and as such, it cannot be used as a parameter to a CORBA object. This caused a problem because we wanted several CORBA objects to use the same database connection, and thus needed to transfer the connection as a parameter.

Another issue to take into account, is that CORBA objects may reside in different processes. This is important to notice when objects contain non-data attributes - like an open socket or file descriptor. So even if we could make an IDL description of the database connection, it could not be passed across a process boundary, because it contains a connection to the database.

The solution was to introduce a single class by collecting all methods requiring a database connection for querying and manipulating the database. By designing systems with this kind of characteristics, we recommend to consider the connections to external resources.

5.1 Experiences with Oracle Application Server

Eventhough, we knew little about CORBA it was very easy to add CORBA functionality to our Java classes. This was due to the fact that Oracle has hidden the CORBA layer within the Application Server. When a Java class should be implemented as a CORBA class we needed to write the interface specification for this class. The interface is a Java interface that extends a special JCORRemote interface that Oracle has defined. Then all we had to do was to deploy our Java class and interface specification to the server in order to make it a fully functional CORBA class.

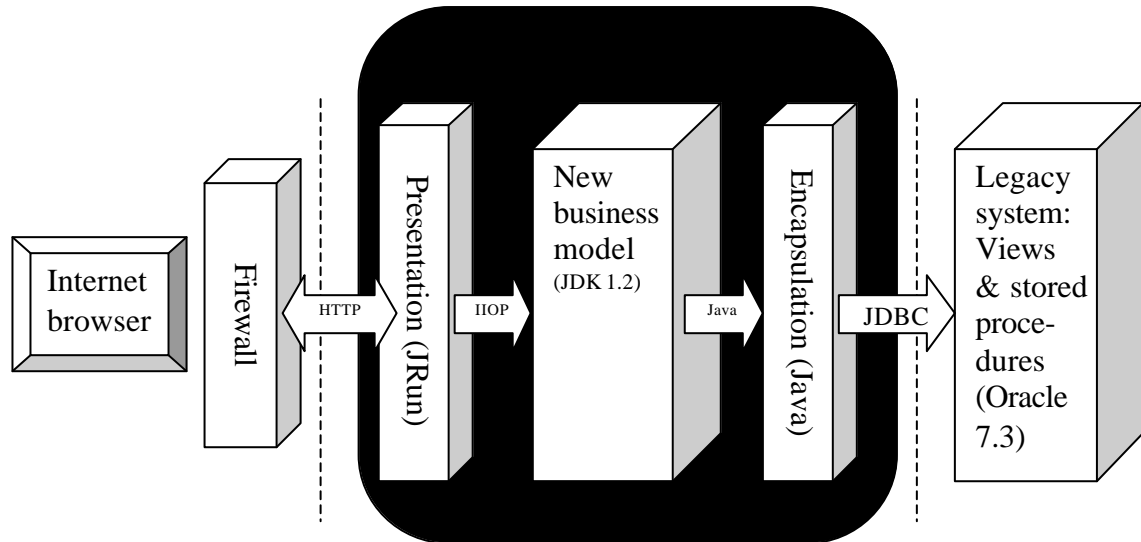
However the platform caused us several problems.

- In order to implement the Singleton pattern [5] for the manager classes we needed methods for storing and retrieving object references. A name service will usually be used to implement the methods but OAS does not support CORBA Name Service properly to be used for this. Other implementation techniques also failed and we were not able to find a solution that worked in OAS though it was crucial for the implementation.
- OAS has its own approach in support for CORBA functionality. This means you cannot use a lot of the existing literature and examples on Java and CORBA like in [6]. You have to use the CORBA documentation from Oracle and we found several situations where it wasn't sufficient.
- During the deployment process of Java classes to CORBA classes on the server, we experienced that OAS failed to deploy the classes when the number of classes exceeded a certain number. The reason seems to be a memory error, and we had to split our Java application into several applications on the Application Server in order to solve the problem.
- The automatic IDL generation in OAS reports very little on errors that might occur during compilation. This makes it very difficult to detect compilation fails.
- It takes a comparatively long time to deploy even a "small" application in OAS, which makes the edit-compile-test cycle a very heavy process for the developers.
- Finally we experienced a bootstrap problem, because some CORBA helper classes are generated on the time of deployment. The problem is that you need the helper classes when you compile your Java code i.e. before deployment.

The problem in implementing the manager classes as Singleton classes prevented us from getting the system to run with OAS. Though a solution probably exists, we were disappointed not to be able to find it within the project period. Thus we were forced to switch the implementation platform in order to get the system running.

5.2 Switching the ORB

Because of the problems experienced with OAS we decided to switch the implementation platform. The new platform was based on public available tools that we could download from the Internet and is illustrated below:



We decided to use the built-in ORB from JDK1.2 [7] to implement the business classes. We also decided to implement the presentation layer as plain Java servlets, which are Java classes that dynamically generate HTML pages for a web server. We selected JRun 2.3 from Live Software Inc. [8] to handle the servlets. JRun is a Java servlet (and JavaServer Pages) engine. It also includes a built-in Web Server that we used to test our application.

Moving the developed code from OAS to the new implementation platform went smoothly though it required code changes, primarily eliminating OAS specific details. The changes made the code more conformant to traditional CORBA/Java development and involved the following:

- We had to write all the IDL ourselves and generate it to obtain the Java classes. We used a trick to catch the temporary IDL files while OAS was deploying the application. These IDL files were then edited to exclude OAS details, and then generated for Java. In other projects the IDL design is an important activity and our OAS trick did not give the most positive solution, but the solution worked!
- The creation of CORBA objects was different. The new code was more simple and should only make a new Java object instance, and connect it to the ORB. We were pleased to find that these changes were isolated to the factories because we used the Factory Method pattern (see section 4.3).
- The Singleton pattern was implemented using the CORBA Naming Service.
- The presentation layer was completely rewritten to Java servlets. This may sound like a major task but the calls to methods in business classes and the generated HTML code were the same and could thus be plugged into the servlet code.
- Minor changes were needed like extending the implementation classes from the generated base classes.

In general these changes were caused by our initial choice of OAS. We believe fewer changes are required to switch between other ORB implementations that conform better to the CORBA standard.

One benefit from an application server like OAS is that it includes facilities to set up multiple applications and configure how they shall run in different processes. It is designed to handle large applications and optimise performance for the created objects. No equivalent tool is delivered along with JDK and we had to implement this ourselves. A simple server program was developed so that all objects were created within this server process.

We found that debugging was a lot easier on the new platform because we could see the output from the server process.

6. Conclusion of experiences

We have introduced a good method to extend legacy systems with new functionality. The use of object-orientation made it relatively easy to build an encapsulation layer on top of the legacy system. We believe the principles can be used in general.

Eventhough, we had little knowledge of this technology we benefited of being mentored by an external resource. We also had success with the use of patterns.

CORBA and Java give a fine platform to implement the architecture we decided to use. CORBA is almost hidden to the client and partly for the server application too. However, in the implementation design you must be careful only to pass CORBA defined data types in calls to CORBA objects. A consequence seems to be that the design must introduce one monolithic class with methods for all the functions requiring a database connection because a JDBC connection is a native Java defined class. We believe our observation is general for similar applications based on Java and CORBA.

We were disappointed with our experiences with the CORBA support in the current version of Oracle Application Server. The Singleton problem actually prevented us from implementing our design on the platform. A key problem is that the product tries to hide CORBA too much for the developer. This makes it difficult to use traditional CORBA techniques and existing literature. We strongly believe that it is better to use tools that really conform to the standard.

We were pleased to experience that switching the ORB platform required very few changes in the code. Having spent more than 600 hours trying to get the system running on OAS it only took two persons one day to switch to the ORB in JDK1.2. It really proves the benefits from using standards like CORBA!

Performance considerations had low focus in our project. The use of the new system will not be very intensive and thus performance is not very critical. We expect the implementation will perform adequately but in a mission-critical system you will probably have to spend extra time checking for unnecessary communication overhead between different objects.

7. References

- [1] Centre for Object Technology, www.cit.dk/COT.
- [2] RAMBØLL Information Technology, www.ramboll-it.com
- [3] Allan. R. Lassen. *WorkSAFE/IKIS empiri*. COT/4-09-V1.0. (Internal working paper in Danish).
- [4] Johnny Olsson, Lisbeth Bergholt, Aino Cornils. *Mønstre - en indføring i analyse-, design- og arkitekturmønstre*. COT/4-07-V2.1. (In Danish).
- [5] E. Gamma, R. Helm, R. Johnson & J. Vlissides. *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1994.
- [6] R. Orfali, D. Harkey. *Client/Server Programming with JAVA and CORBA*. 2nd edition. Wiley, N.Y., 1998.
- [7] Java Development Kit Software, <http://java.sun.com/products>
- [8] Live Software Inc., www.livesoftware.com