

Understanding The Key Elements For Successfully Building CORBA Into a Distributed Telecom Network

Andres Koch, dipl. El. Ing. HTL, M. Math
Object Engineering GmbH, CH-8142 Uitikon-Waldegg
Switzerland, <http://www.objeng.ch>
E-mail: akoch@objeng.ch
Paper presented at

Evaluating The Business Potential of
CORBA
In The Telecommunications Industry

Monday 15th, Tuesday 16th & Wednesday 17th June 1998
The Cumberland Hotel, London

Author

Andres Koch is the founder of Object Engineering GmbH, a company which is involved in engineering legacy systems to migrate and integrate them into more modern distributed environments. He received a Bachelor of Electrical Engineering from the Engineer's School of Technikum Rapperswil ITR and a Master of Mathematics (Computer Science) from the University of Waterloo, Ontario Canada. He also teaches part-time postgraduate courses in studies of software engineering at Hochschule Rapperswil (HSR) covering Distributed Computing, Object Oriented Analysis and Design.

Abstract

Object oriented technology and distributed object technology have lots in common, even though the second implies even more complexity to it. This paper wants to emphasize the importances of some key elements when developing CORBA based systems or migrating legacy applications to this architecture.

The application used as the case study in this paper is part of a larger Swiss Telecom system which contains an MVS-based host for administering subscriber contracts and data. The application is a new front-end to manage all the subscriber contracts. There are approximately 3000 PC users in 17 different locations in Switzerland. The migrated system is a special decentralized system, which used to function as a terminal server. The migration has been in progress for about four years, and it is continuing as more legacy components are migrated. The paper focuses on the latest stage, where CORBA was used to connect the GUI-driven PC application to the decentralized servers and to the central host in a typical 3-tier architecture.

The described project awarded first prize at COMDEX Internet and Object World Frankfurt 1997 for "Best use of object technology within enterprise or large system environment".

The Key Elements

Object Orientation As The Base

By using the object oriented technology one will experience that:

- there is improved and systematic support of modularity by methods and tools.
- inheritance is not always necessary (see Gamma/Design Patterns)
- change management becomes a very important issue.
- the need of a class and interface repository is viable.
- system and software architecture for the enterprise becomes a very important issue

And on the other hand some well promoted advantages become myths, because:

- more parts to the whole exist then with classic software development (Monolith)
- reuse in large coperations is a myth if not strongly supported by policies and if it is not cared for.
- lack of information flow in enterprises makes it difficult to get the components (classes) known.
- typically a component becomes “unreusable” within enterprise after two years if not maintained.
- missing component use due to unknown existence, bad interfaces or documents, badly designed or hard to understand.

What are we able to learn from this story? That object oriented technology is a very good thing but it has to be supported by some organizational process. It probably helps if we try to learn from other trades like machine construction where one has to deal with lots of components too (parts list).

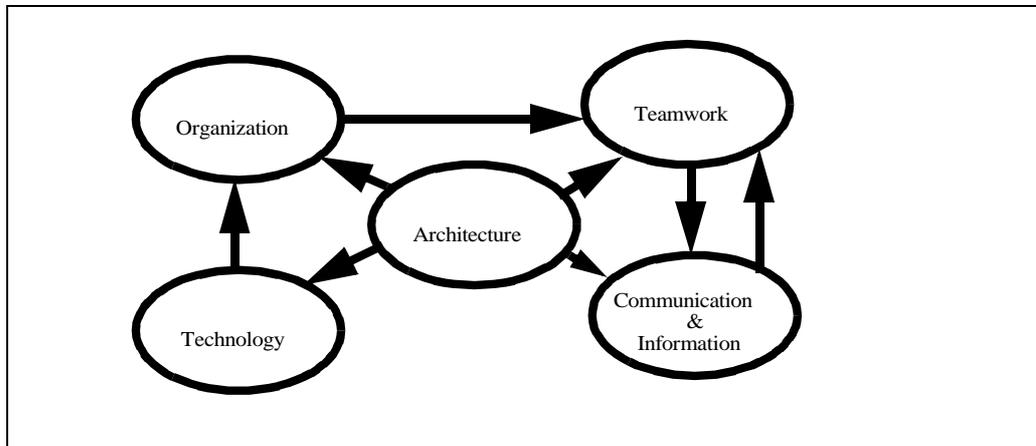
Key Elements Of Distributed Object Technology

What about objects which are deployed over the net? Most of the issues above are true for distributed object technology, but even to a higher degree. If you would say while using CORBA-middleware that you have a flexible distributed system it would be the same myth as if you would try to say that while using C++ that you have an object oriented system.

Additional Issues in a distributed environment are:

- Physical separation and distribution of application parts
- Real concurrency
- Need for coordinating the interfaces of the components (Repository)
- Flexibility also in the area of deployment and operation
- Challenge for development AND deployment.

The question is how to meet this challenge and how to find a solution to it. Experience in projects in this field show that the following pattern is a workable solution:



The main elements are:

- Architecture
- Organization
- Teamwork
- Information & Communication
- Technology

The object management architecture by OMG is a good starting point. The problems about reuse and the mass of parts are very similar using OMA and CORBA as a base architecture. It should forward the *form* and the *tactics* how one can meet the target with available resources. But there is need for a simple, understandable, known and made in time architecture which is well documented. It is very much recommended that you have one architect since groups tend for compromises. The architect should then also follow up as a quality assurance function that his/her architecture is consequently executed by the developers. By the way would you build a house for two million dollars without an architect? And why would you employ a software project for the same cost without an software architect?

This has to be supported by an organization which evolves from the architecture (product oriented) since the architecture gives the guideline how to distribute the components into the team. Thus the parts developed as subsystems or components have to be known and accessible to the other team members. And the interoperability is done over well defined interfaces of the components. This is finally an organizational matter and the utilization of the human resources as the biggest cost factor are very dependent on this.

Needless to say that the team is the biggest factor to the success or failure of a project. It needs just one person to crash the project but it needs more than one person to make it a success. The character of a distributed application or an application separated in components has the advantage that it can be easily distributed among the team members. A team size of three to five people has shown as the most efficient size of the group. On the other hand these teams have to be coordinated and this is done by the project leader and the architect. Each team has to have a clear statement of their task, responsibility and competence. Check the quality of your team by asking yourself if you would undertake an expedition to the north pole with the members of your team

It can easily be understood that communication and information flow are as essential to the used technology as for the organization and the teams. Since more self contained groups exist their experience which might

be common to others have to be communicated. The intranet forum home pages is the ideal tool for this (if you are leaving the fancy parts out) but also walkthroughs organized as workshops are very useful.

There might be some astonishment by the reader, why technology is left to the end. Well it is a key element but the one with the least importance in the list. Sure the architecture will choose the technology, but if it is not supported by a good organizational pattern, a brilliant team and a good information and communication flow then your project is at stake. But using CORBA based architecture and tools is very helpful and can be a factor for success for your project and most of all for the future of your company too.

Migration

After isolating the key elements above we can now look at a process used to migrate any system to a better usable system.

Migration is derived from the Latin root “migrare”, meaning to change one’s abode. Successfully migrating a legacy system requires more than just a “change of abode”, it requires a detailed evaluation activity such as the “mining process” described by Mowbray. During this process, the system architect must decide which system components to preserve and which components to replace. Even though many data processing systems have similar processes and data structures, the evaluation for each system is unique. Migration may be required for a variety of reasons: new technology, a changed overall architecture (e.g. Client/Server) or a new development paradigm.

Analysis of the original system is absolutely necessary. It is usual to differentiate between the migration of **Legacy data** and the migration of **Legacy algorithms and processes**. Legacy data must almost always be preserved intact. The conversion process must be exact and precise to ensure that the data (often the heart of the business process) is accurate. Algorithms and logic can often be replaced, but the existing algorithms and processes should be examined carefully. There is often a great deal of hidden information in the algorithms - hidden due to a lack of documentation or due to an overwhelming volume of documentation. A careful evaluation and analysis is vital to make the migration as smooth as possible.

Migration requirements can include:

- “Big-Bang” replacement - this is seldom desired nor can it be risked.
- Migration cannot interfere with day-to-day operations.
- The system must remain available during the migration.
- For practical reasons, the old and new systems must operate in parallel (e.g. 3000 systems take time to replace)
- Users should only notice positive changes.
- The migration process is often started late and has to be finished under tight, high-pressure deadlines.
- Performance constraints must be avoided
- Costs should be kept within budgetary limits.
- System maintainability should be improved (or regained in the case of obsolete components).

Software engineers who can manage the migration process with technical, administrative, and human communication skills are one of the most important factors for the success of such a project.

Case Study: The project

The migration process has been underway for about four years at Swiss Telecom. The system consists of a central information application running on an IBM mainframe under MVS with decentralized subservers in 17 regional branches serving approximately 3000 user terminals. The remote servers are used for terminal multiplexing and communication processing. The application manages the connection and inventory data for telephone and service subscribers.

Many changes were made to the original system over the past 15 years to respond to new user requirements and business requirements. As a result, the application became rather complex. The new system architecture was systematically designed to ensure that the migration process was possible within the available budget. While the original system consisted almost entirely of proprietary hardware and software incapable of interoperation with other platforms, the new system is constructed from standard, widely used components. These include PC platforms with Windows 95, a CORBA environment, and multi-platform accessibility over CORBA-middleware Orbix and TCP/IP.

Original Scenario

The original system architecture included a central mainframe with the MVS operating system. The decentralized and regional systems were Series/1 (S/1) processors running under a special multitasking operating system (RPS). The remote systems were connected to the mainframe using the SNA/LU0 protocol over serial lines running at 9600 baud. Up to 32 dumb display terminals were connected to each of these S/1-systems.

To minimize the traffic on the relatively slow serial lines, only the form data was passed in messages between the host and remote systems. Screen layouts and masks were held on each server and merged with the data in the messages from the host. There were approximately 300 masks in use for each of the three languages supported (900 masks in total). A small part of the application software ran in each decentralized system to provide user comfort functions, data integrity tests, and access control based on downloaded user profiles.

System replacement or migration became necessary because:

- The remote subservers was limited to a maximum of 32 terminals
- The systems were no longer being manufactured
- Hardware maintenance was suspended shortly after manufacturing ceased
- The personnel capable of performing software maintenance were increasingly hard to find

Ignoring these problems would have exposed the organization with excessive risk.

System Migration

Phase 1

In phase one of the migration, the proprietary display hardware was replaced with first-generation PCs. The PCs were connected to another subsystem using the CTOS operating system to provide additional application and office functionality.

Phase 2

In phase two, proprietary hardware in the decentralized systems was replaced with much more powerful, modern, RISC-based systems running Unix. By a fortunate coincidence, software was available that ran on Unix and RS6000 simulating the S/1-hardware under RPS. This provided a method to make the system interoperable with other systems. It was also possible to connect the CTOS-systems by using sockets over TCP/IP. The host was still connected over SNA/LU0 but this protocol was now tunneled within the TCP/IP protocol using the cooperation network instead of serial lines. The software that ran on RPS (written in S/1 Assembler and PL/1) ran unchanged on the VS/1 (Virtual Series/1) emulation software. The software actually performed better (by a factor of 5) due to the superior performance of the new generation of RISC hardware.

New interoperability components were tasks running directly under Unix and connected to the VS/1 using the interprocess communication facility (Unix message queues). These tasks were written in 'C' (later in C++) using object-oriented design by Booch and a CASE tool that supported this method.

At this point, the major problems had been solved and the risk of interrupted operation had been removed. However, the implementers knew that this could not be a permanent solution. The first generation PCs were replaced by standard PCs using VT220 terminal emulation to connect to a specially designed subprocess (TRM) on the VS/1. Even so, the user interface was still character oriented and the new "Windows-Wave" could not be used by this application.

Phase 3

In phase three there was considerable pressure from the users to implement the system on a standard Windows[®] platform. The host application's user interface was redesigned and implemented using MFC in MS Visual C++.

Because none of the original host development personnel were available, there was great reluctance to make changes to the host software or the VS/1 software. To solve this problem, a "facade" was designed which "wrapped" the VS/1 and host system and provided a logical representation of the data to the client application running on the PC. Since the interoperability and connectivity had to overcome system and network boundaries, the designers decided to use CORBA-based middleware, Orbix[®] by IONA.

The typical character-based user interface presented by the host was abstracted and presented to the client as a network database data record. The wrapping was made in a CORBA-based data interface service which was located on the VS/1-server, based on the components already used in the previous steps of the migration. This approach also provided a foundation to replace the entire VS/1 and its hard-to-maintain software at a later date. The user interface could remain unchanged while the service implementation (including the VS/1) is redesigned, streamlined and replaced.

Phase 4

Finally the last step currently under way is to replace VS/1, the legacy component which is the biggest performance and maintenance bottle neck. This is done by using a CORBA-middleware (OrbixMVS) on the mainframe. This includes a service which acts as a Gateway to the IMS-System (IMS-Gateway) and provides a CORBA-based interface on the other side.

VS/1 and the SNA/LU0-facility is removed. The data interface service (DIF) is extended going through the CORBA-track directly to the IMS. More application logic which has been hosted by the VS/1-programs is moved into the data interface or is covered by the host application. Due to the object oriented design of the DIF-software, lot of the existing work can be reused. The interface of the DIF stays unchanged and thus the

frontend application have not to undergo any change.

A prototype built based on this approach showed that we were able to decrease the response time from around 6 seconds to around 2 seconds. The maintainability is improved too (less legacy components).

The step after this should then include to migrate the mainframe application into a business object model.

Summary

Although the original architecture required compromises to be made during migration, overall the project was a success.

Due to dependencies of the client (data), it was not possible to completely encapsulate the application architecture on the mainframe. This made it necessary to provide version control. A major restriction was that the existing host application had to remain unchanged. The additional abstraction layers that this required increased the system complexity. This complexity was tolerated for this version, but it must be reduced in a subsequent redesign.

Using visual C++ on the client part resulted in a rather “heavy” client. A light-weight client should be considered as an alternative, perhaps using a web browser with Java and with a Java-based ORB.

Although the general pattern followed in this migration may be applicable to other systems, each project requires individual analysis to determine the optimum path of migration.

As shown by this project, CORBA tools for middleware can be used in real-world applications without problems. In general, developers spend too much time in academic discussion of the tools instead of using them and concentrating on the design of distributed applications. The key to the success of this project was the definition of an overall architecture and an interface framework for CORBA-based applications within the IT-department of Swisscom. Good communication between the development teams for the host, server, and PC software was vital for a project success.

During development, the teams must not forget the importance of the maintainability and ease of operation of the distributed applications. A good mixture of pragmatic solutions and long-range planning is invaluable, since there are problems to solve in the short, medium and long term.

- [1] G.Booch, "Object Oriented Analysis And Design with Applications", The Benjamin/Cummings Publishing Company, Inc. New York, 2nd Ed, ISBN 0-80535340-2, 1994
- [2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley Prof. Comp. Series, ISBN 0-201-63361-2, 1995.
- [3] Mowbray Th, Zahari R, "Essential Corba: System Integraton with Distributed Objects", John Wiley&Sons, ISBN 0-471-10611-9, 1995
- [4] Object Management Group, "Object Management Architecture Guide", Rev. 2.0, Sept. 92, John Wiley&Sons, Inc, New York, OMG TC Document 92.11.1, ISBN: 0-471-58563-7.
- [5] Object Management Group, "The Common Object Request Broker: Architecture and Specification", Rev. 1.1, John Wiley&Sons, Inc, New York, OMG Document 91.12.1, ISBN: 0-471-58792-3.