Implementing Business Objects:

CORBA interfaces for legacy systems

Thomas Grotehen
University of Zurich
grotehen @ifi.unizh.ch

René Schwarb
SYSTOR AG
Schwarb.Rene@ch.swissbank.com

Abstract
In 1991, the OMG (Object Management Group) defined an architectural framework (OMA-Object Management Architecture) as a milestone in realizing the vision of distributed object-oriented computing. This paper describes an experiment designed to examine whether a OMA CORBA (Common Object Request Broker Architecture) implementation [OMG92, Mza95] can be successfully employed in the existing information technology environment of a bank. It provides an overview of both the benefits and the problems involved and an outlook on future technology developments in this area.

1. Introduction

This paper provides an overview of an experiment using a CORBA implementation. The experiment [Gro94] was carried out to illustrate that CORBA implementations can be used to define a layer of business objects above existing legacy systems in a large scale financial environment. The experiment included 11 servers that allowed 12 clients to access data from legacy systems, such as product and customer data, document information, UNIX-mail, account information and data dictionary entries.
The server applications run on UNIX machines and access other UNIX, CTOS and MVS machines. The server applications can be accessed by UNIX, Windows 3.1 as well as Windows NT clients. They use IMS and DB2 transactions, a variety of interfaces to these

transaction systems, Sybase Open Servers, a document generator and an interface to an electronic document management system.

This document will provide an overview of the existing legacy environment, the implemented client and server applications, and the development tools used including CORBA. In addition, we will illustrate the manner in which this evaluation confirmed our expectations with regards to the benefits derived from this architecture (integration of legacy systems, use of heterogeneous development tools and programming languages) and also show what problems can occur when using this approach.

2. The environment

2.1. Underlying legacy systems

The underlying systems represent a mixture of computing environments implemented over a period 20 years. Most data is managed by IMS database systems running on MVS. Some data is replicated on DB2 database systems also running on MVS. This data is accessed by many RTB (Real Time Banking) transactions running on IMS/DC and CICS.

A number of existing client applications run on the CTOS operating system. There are various interfaces from CTOS, DOS, OS/2 and UNIX to the transactions. In addition, interfaces exist between UNIX and CTOS, allowing access to the transactions. Furthermore, there is a DDCS (Distributed Data Communication Services) interface connecting UNIX to the MVS DB2 databases and a specific interface provides connection to an important subset (CIF-Central Information File) of the RTB transactions accessing customer and capital data.

2.2. Current development systems

Sun Solaris and the SYBASE database management system, were chosen for the server platform. The client platform was determined by banking decisions which require seamless integration of Windows applications such as MS-Word and MS-Excel with banking applications developed in-house. Therefore, client applications must run on Windows, while accessing server functionality on UNIX. The client development environment includes ART*Enterprise, Visual C++ and Enfin and the servers have been developed in C++ using several C++ development tools. The server access underlying systems run on UNIX, CTOS and MVS.

3. CORBA business objects

One of the main problems in current projects is the use of underlying legacy systems. There are various approaches to accessing legacy data. In most cases there exists an interface function. In our experiment application developers view legacy functionality through business objects. The approach consists of three steps:

1. analysis of business objects

2. specification of interfaces to business objects in CORBA-IDL (Interface Definition Language)
3. implementation of interfaces

The analysis of business objects is probably the most complex step. There are a variety of approaches in this area, however, most approaches do not take into consideration analysis of existing systems. We opted for the SOMA methodology [Gra94]. The result is a set of business object specifications (class cards). These have been mapped to existing transactions. Fig. 1 illustrates two simplified class cards.

Class **Product**	*Superclasses*
Attributes *string date;*	
Public Methods **string code ();** **string condition ();** **...**	*Collaborators*
Private Methods **string date ();**	*Collaborators*
Rules	

Class **Product Factory**	*Superclasses*
Attributes *string date;*	
Public Methods **Product capture Product (in string code //Productcode in string date) //Datum**	*Collaborators*
Private Methods	*Collaborators*
Rules	

Fig. 1 Simplified class cards

The definition of interface specifications is straightforward. Each class card results in an interface. Class card public methods and attributes are mapped to attributes and operations of the interface. The following are the interface specifications derived from the class cards in Fig. 1

```
interface Product{
        string code ();
        string condition ();
        ...
};

interface ProductClass
{
        Product captureProduct (in string code  //Productcode
                                in string date)  //Datum
};
```

Fig. 2 Interfaces derived from class cards

This interface specification is used on both the client and server portion of the application. On the client side, the IDL compiler generates code (IDL stubs) which can be transparently used by client programs. (e.g. in a Visual C++ program)

The final step is to implement the server interface. In most cases we have used existing functional interfaces in a very simple architecture (one to one relationship between business objects and transactions).

4. The experiment

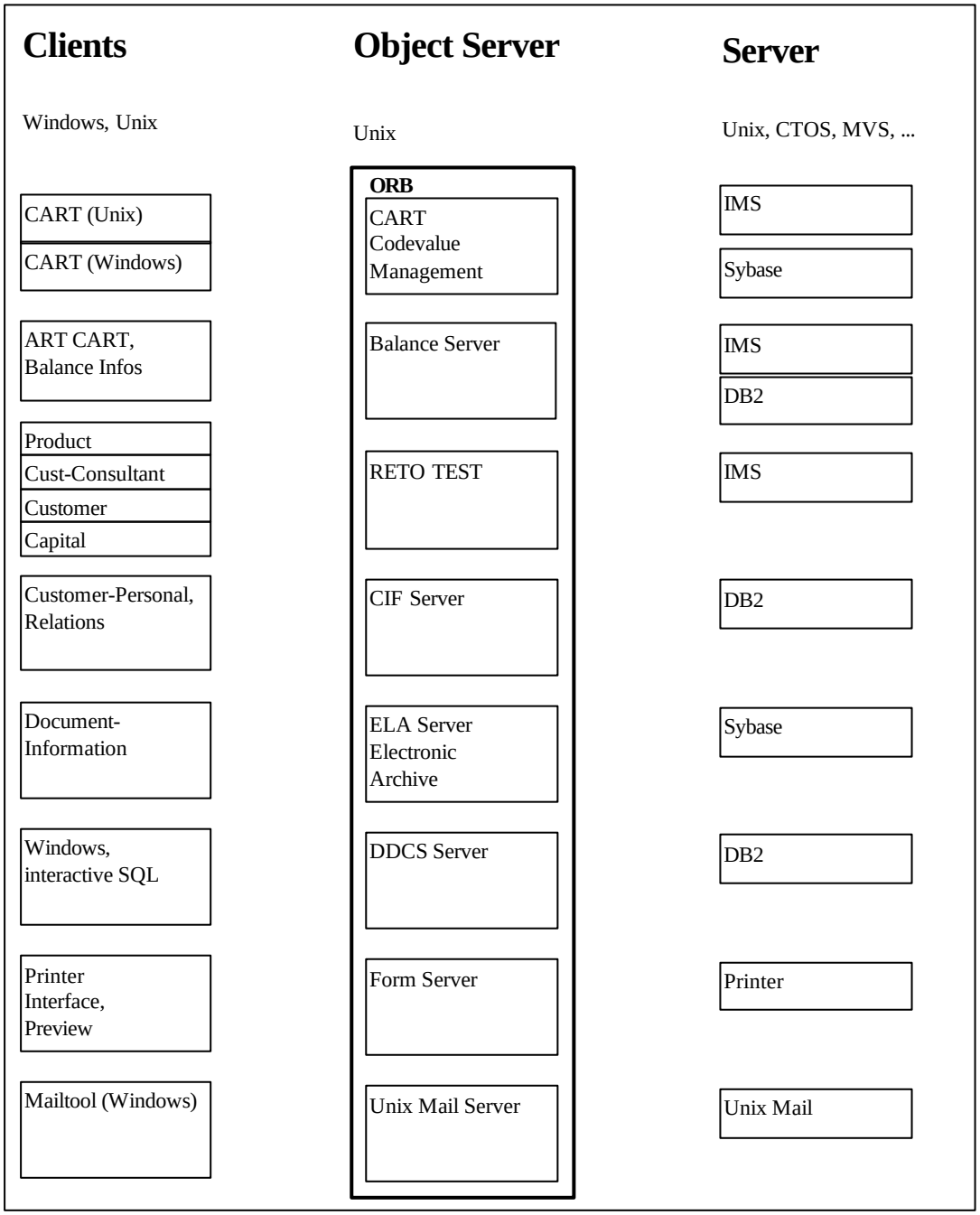

Fig. 3 Clients, Object Servers and existing Systems

Most implemented applications consist of three parts. A very lean client, representing a user interface. The client runs on Windows and accesses a server via an ORB (Object Request Broker). The object server runs on Solaris and uses additional resources such as IMS transactions (minor modifications have been made in the legacy environment but they are not discussed in this paper).

Fig. 3provides an overview of the resulting system:

CART(Codes at Runtime) [KJM93] is an existing application developed in C that allows access to IMS data dictionary information at runtime. Certain CART functions access IMS databases, others already use the Sybase database system. Our implementation provides a seamless integration of CART-services with UNIX- and Windows-C++ applications.

One of the CART UNIX client applications is the ART client. It has been developed to illustrate that client applications developed using the ART*Enterprise environment can use the functionality provided by ORB-servers. An ART-CART client simply integrates the CART functionality accessing the CART server for example to display codes. Once the integration of the CART-functionality was illustrated, a balance server providing customer balance information using two IMS-transaction was implemented.

The RETO-test application provides a small section of a retail tool. The RETO clients provide information on customers, products and customer capital. The RETO object server uses the corresponding transactions to provide data and transactions in the form of C++ classes.

The CIF (Central Information File) client provides access to CICS transactions using a Sybase network gateway and delivers data representing customer information and customer relations.

ELA (Elektronische Ablage (electronic filing)) is a product running on UNIX that allows electronic scanning, storage and access of contract documents. This access is required by several other systems. The server provides an interface to this system on UNIX that can be used by UNIX and Windows Clients in the form of C++ classes.

The DDCS server shows the implementation of an interactive Windows interface to a DB2 database using a DDCS interface. The clients allow users to specify SQL queries and convert them to DB2.

The Formula application consists of a formula server that accesses formula scripts that create documents using dynamic data from the client applications. More than 700 scripts are used to prepare customer output. The client first requests to prepare the document. The client can then retrieve a postscript file that can be viewed with a PostScript viewer or printed on the client's printer.

The Mailtool server encapsulates the UNIX mail command. We believe that this is a good example of a CORBA facility. The Windows client is a Windows Mailtool that allows Windows users to send and receive UNIX mail. The C++ classes allow Windows programmers to use the mail functionality from within their applications.

5. Benefits and problems

5.1. Benefits

In comparison with other approaches, this approach provides a variety of benefits. For example, in a two tier architecture in which legacy functionality is directly used in client applications most of the functionality concerning legacy data, such as data conversions is implemented on the client and as such is tightly coupled with the user interface. This makes it difficult to provide reusable functionality since decoupling from the user interface would be required. It also makes it difficult for other clients to reuse the functionality at runtime since code changes require recompiling and subsequent changes.

A layer of object servers makes it possible to implement new functionality separated from the user interface, and clients can use this functionality using a variety of programming languages and operating systems.

The interface to the classes is described by IDL, i.e. independent from a particular programming language. The CORBA implementation maps the IDL-descriptions to different languages supported by IDL on a variety of operating systems, making mapping reuse simple.

Furthermore, applications within different projects can use this approach to describe their interface. This results in uniform interfaces, even if the applications are developed in different programming languages. The CART application, for example, is implemented in C. Other applications developed in C++ use a set of C++ classes which are generated by the CORBA implementation to access CART.

The interface description of existing object-servers can also be used as a design documentation for a new application. Therefore, reuse is not only supported at run-time but also at the design level. In an environment where multiple applications co-exist that communicate with each other using ORB mechanisms, the development of a new application now begin with a set of uniform interface descriptions. These interface descriptions can be integrated into the design of the new application. The interface descriptions can then be used to generate the interfaces to the corresponding (existing) servers.

In summary, the main advantages of this approach are as follows:

- providing support for decoupling the user interface from the server functionality
- simple mappings of legacy functionality to object types
- uniform (standardized, object-oriented) interface descriptions
- easy reuse of applications and design specifications in a heterogeneous environment

5.2. Problems

Early CORBA adopters may face a number of problems. The technology is still in its infancy, especially as object services used by the ORB are still in a standardization and

implementation phase. An example of such an object service is the transaction service. This service has already been adopted as OMG technology but not included in CORBA implementations. Therefore, applications requiring concurrent access to shared data must provide their own mechanisms for handling distributed transactions.

Services such as persistence service, concurrency service as well as recovery service are mechanisms for providing object-level transactions with ACID (Atomicity, Consistency, Isolation, Durability) properties.

Additional problems concern high availability and fault tolerance of distributed applications. These requirements have been addressed by toolkits for distributed computing such as Isis [BvR94]. Unfortunately, only a few CORBA implementations take these requirements into consideration. An example is recovery from failures in server processes. If a client has already communicated with a remote server and the server application fails, most current CORBA implementations do not provide any support to the client application. The application must be written such that it tries to recover connection to the server (which is very complicated) or the problem is left to the end user. Possible solutions to this and other problems with high availability and fault tolerance can also be achieved by combining CORBA implementations and network toolkits [OI94]. The key idea behind the Isis toolkit, for example, is that services are based on groups of processes running on different machines (more or less transparent to the user of the service).

Other problems concern network security requirements, such as the secure transmission of requests and data within a distributed system and the authentication of a request originator. If an authorized client application communicates with a server, an intruder can disconnect a client and misuse the already established connection to the server, pretending to be the client that is authorized access the server. These and other security-related problems have been addressed by mechanisms such as secure RPC (Remote Procedure Call) or Kerberos [Koh92]. Unfortunately, only a few CORBA implementations use secure messaging mechanisms. The application programmer thus has to deal with all security problems. Fortunately, most current CORBA implementations provide for convenient access to their mechanisms. Furthermore, basic security mechanisms and algorithms that fulfill a variety of security requirements exist and can easy be integrated into the applications.

6. Future developments and conclusions

In the future we must consider the following developments in the CORBA architecture:

1. CORBA implementations will be integrated into operating systems
2. CORBA implementations from a variety of vendors will be interoperable. (CORBA 2.0)
3. Microsoft has endorsed CORBA, with the objective to integrate CORBA and OLE with COM (Common Object Model). CORBA implementations with an OLE bridge are already available
4. Implementations of CORBA services will extend the functionality of CORBA implementations (COSS 1-4)

5. CORBA Facilities will provide for a set of standardized services to be used by applications
6. More dynamic and flexible tools (e.g. Java [SUN95]) will be available

This will lead to major changes such as changes in development tools, and the level of functionality provided. However, none of the changes require alteration to the basic concepts behind object-oriented technology. Therefore, the notion of an *interface for an object* is stable. CORBA-IDL is merely a simple description language for such interfaces. Furthermore, uniform development environments in large companies are unlikely for the near future. Heterogeneity will be the rule and a stable platform is an essential requirement for such enterprises. Therefore, building business objects based upon CORBA-IDL and using CORBA as the backbone of an enterprise architecture becomes is a reasonable approach for the integration of complex systems.

References:

[BvR94] Birman, K.,P., Van Renesse, R.(eds.): Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press, 1994

[Gro94] Grotehen T.: Evaluation: CORBA-Architecture, Swiss Bank Corporation, Domestic Division, Internal Report, 1994

[Gra95] Graham, I.: Migrating to Object Technology, Addison-Wesley, 1994

[KJM93] Kannappel, R., Jordan, J., Mühmel, J., Negroni, M., Oswald, U., Schwarb, R., Studer, C., Toman, J.: CART-Codes at Runtime, Requirements Planning, Final Draft, SBV Basel September 1993

[Koh92] Kohl, J.B., Neumann, C.: Kerberos V5 Specification. Internet -draft, September 1992

[Mza95] Mowbray, T.J., Zahavi, R.: The Essential CORBA, Systems Integration Using Distributed Objects, Addison-Wesley, 1995

[OI94] IONA Technologies Ltd and Isis Distributed Systems, Inc.: An Introduction to Orbix+Isis, IONA Technologies, 1994

[OMG92] Object Management Group: Common Object Request Broker Architecture. Framington, MA, 1992

[SUN95] Sun Microsystems. Java - The Language. Sun Microsystems Documentation 1995