18

CORBA access to telecommunications databases

P. Porkka and K. Raatikainen
University of Helsinki, Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland.
Telephone: +358 9 7084 {4677,4243}. Fax: +358 9 7084 4441.
E-mail: {pasi.porkka,kimmo.raatikainen}@cs.Helsinki.FI

Abstract
Distributed object technology, CORBA in particular, will be the basis for the next generation of telecommunications software. Intelligent Networks Long-Term Architecture, Telecommunications Management Network, Telecommunications Information Networking Architecture are all based on object technology. The Telecommunications Task Force of OMG is actively working towards CORBA-based solutions.

In this paper we present how the standard CORBAservices specified by OMG can be used to provide access to telecommunications databases. The challenging task is to introduce database objects into Object Request Broker without registering each database object as an ORB object. The RODAIN Object-Oriented Database Adapter (ROODA) is our solution to bring database objects and services for CORBA clients. The ROODA implements interfaces that will provide essentials parts of Persistent Object Service, Object Transaction Service, and Object Query Service as well as the Dynamic Skeleton Interface.

Keywords
Database Access, CORBA, Object Services, Object Database Adapter

1 INTRODUCTION

Databases are important building blocks in modern telecommunications systems. Databases are already used in several areas, such as call connection, number translations, Intelligent Network services, mobility management. Due to the growing use also data value and volume are increasing rapidly. The requirements for database architectures to be used in the telecommunications originate in the following areas: real-time access to data, fault tolerance, distribution, object orientation, efficiency, flexibility, multiple interfaces, and compatibility with other object standards (Taina *et al.* 1996).

Real-time access to data means that transactions usually have deadlines that specify when the transaction must be finished. Fault-tolerance means that the database should, in the practice, be almost always available. The requirement of object orientation is based on the general trend in telecommunications standards: **Intelligent Networks Long-Term Architecture** (IN LTA) (ITU-T 1993), **Telecommunications Management Network** (TMN) (ITU-T 1992), and **Telecommunication Information Networking Architecture** (TINA) (Barr *et al.* 1993) are all based on object technology. In particular, the de facto standards based on the OMG specifications are of crucial importance.

The efficiency requirements that the telecommunications services set to databases are strict. Thousands of short transactions must be answered in a second. At the same time very long transactions must also get resources. The requirement of flexibility arises from the fact that telecommunications databases must support very different kinds of transactions. Multiple interfaces to the database are a necessity. A telecommunications database has different types of users. For example, some of the users want to see the database as an X.500 Directory while others want to see it as an X.700 Management Information Tree. In a near future, TINA object invocation and access based on the OMG CORBA will be required.

In this paper we describe how database access through CORBA can be arranged. In particular, we describe how we will implement a CORBA compliant **Object Database Adapter** (ODA) called **RODAIN Object-Oriented Database Adapter** (ROODA) that provides a CORBA access to the RODAIN Database. In addition to the OMG Object Services called **Object Transaction Service** (OTS) and **Persistent Object Service** (POS), the ROODA provides an IDL interface to an ODMG-93 (Cattell 1994) compliant **Object Query Language** (OQL) as specified in the OMG **Object Query Service** (OQS). The ROODA also allows applications to register their own IDL interfaces to speed-up the database access.

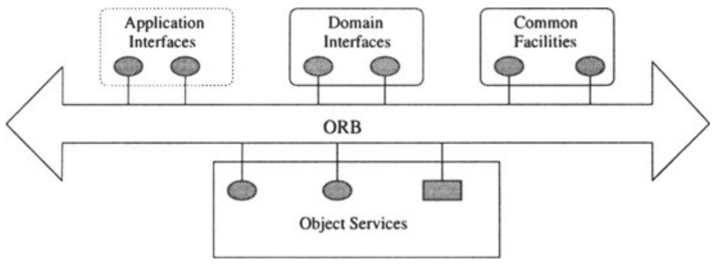

Figure 1 Object Management Architecture.

The rest of the paper is organised as follows. In Section 2 we describe the essentials of the OMG specifications that affect any implementation of ODA: Object Management Architecture (OMA), Object Request Broker (ORB), and Object Adapters. In Section 3 we review two known Object Database Adapters called Sunrise ODA and Orbix+ObjectStore Adapter. Finally, in Section 4 we present the RODAIN OODA and its functionality which includes the Object Transaction Service, the Persistent Object Service, and the Object Query Service as specified by OMG.

2 OBJECT MANAGEMENT ARCHITECTURE

The Object Management Group (OMG) has defined an architecture called Object Management Architecture (OMA) (OMG 1993) that provides the conceptual infrastructure upon which all OMG specifications are based. The OMA has gained the status of the most important de facto standard in the area of distributed computing. In this section we describe the essentials of OMA for database access in distributed computing environments.

2.1 OMA Overview

The key building blocks of the Object Management Architecture are shown in Figure 1. They include Object Request Broker (ORB), Object Services, Common Facilities, Domain Interfaces, and Application Interfaces.

Object Request Broker (ORB) (OMG 1996a), commercially known as CORBA, is the communications backbone of OMA. The ORB transparently provides its clients to make requests and to receive responses using object invocations in a distributed environment. **Object Services** (OMG 1996b) is a collection of services (interfaces and objects) that support basic functions for using and implementing objects. For database access the most important services are the Object Transaction Service (OTS), the Persistent Object Service (POS), and the Object Query Service (OQS).

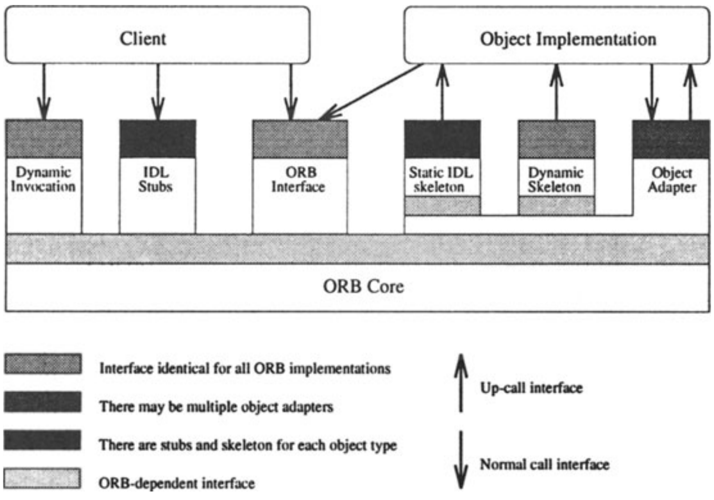

Figure 2 Structure of ORB interfaces [3/CORBA V2.0].

Common Facilities (OMG 1995) is a collection of services that many applications may share but which are not as fundamental as the Object Services. The Common Facilities are divided into two major categories: Horizontal Common Facilities that are used by most systems, and Vertical Market Facilities that are domain-specific. **Domain Interfaces** represent vertical areas that provide functionality of direct interest to end-users in particular application domains. Domain interfaces may combine some common facilities and object services but are designed to perform particular tasks for users within a certain vertical market or industry. **Application Interfaces** while not an actual OMG standardisation activity are critical when considering a comprehensive system architecture. The Application Interfaces represent component-based applications performing particular tasks for a user.

2.2 Essentials in CORBA

The CORBA makes an interface between clients and objects allowing object implementations to be machine and language independent. As Figure 2 depicts, a CORBA client has three primary ways of making a request:

1. The client can use the Dynamic Invocation Interface.
2. The client can use an OMG IDL Stub.
3. The client can directly interact with the ORB through the ORB Interface.

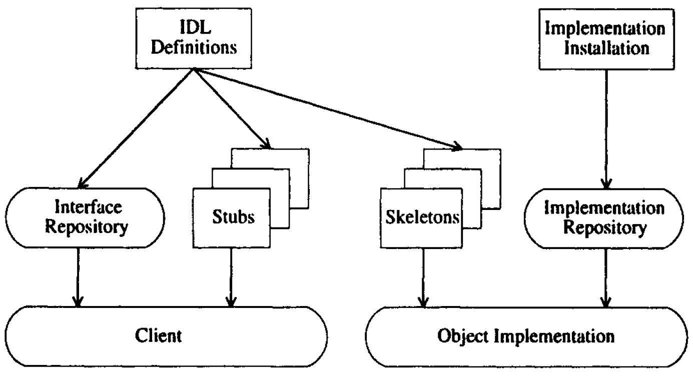

Figure 3 Interface and implementation repositories [6/CORBA V2.0].

Figure 3 shows how an interface and implementation information is made available to clients and object implementations. The interface is defined in OMG IDL and/or in the Interface Repository: The definition is used to generate the Client Stubs and the object Implementation Skeletons. The object implementation information is provided at installation time and is stored in the Implementation Repository for use during the request delivery.

Interface Repository is a service that provides persistent objects that represent the IDL information in a form available at runtime. The Interface Repository information may be used by the ORB to perform requests. Moreover, using the information in the Interface Repository, it is possible for a program to encounter an object whose interface was not known when the program was compiled, yet, be able to determine what operations are valid on the object and make an invocation on it.

Implementation Repository contains information that allows the ORB to locate and activate implementations of objects. Although most of the information in the Implementation Repository is specific to an ORB or operating environment, the Implementation Repository is the conventional place for recording such information. Ordinarily, installation of implementations and control of policies related to the activation and execution of object implementations is done through operations on the Implementation Repository.

In addition to its role in the functioning of the ORB, the Implementation Repository is a common place to store additional information associated with implementations of ORB objects.

Object Implementation is a definition that provides the information needed to create an object and to allow the object to participate in providing an appropriate set of services. An implementation typically includes definitions of the methods that operate upon the state of an object, and information about the intended type of the object.

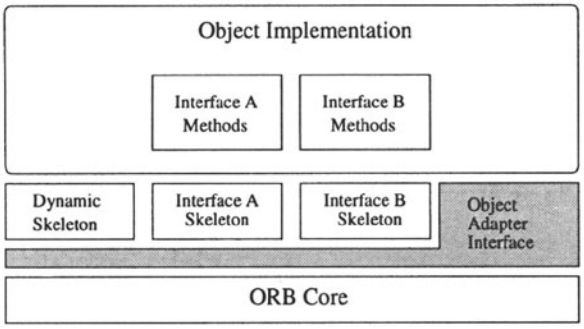

Figure 4 Structure of a typical object adapter [9/CORBA V2.0].

An object implementation provides the semantics of the object, usually by defining data for the object instance and code for the methods of the object. Often the implementation will use other objects or additional software to implement the behaviour of the object. In some cases, the primary function of the object is to have side-effects on other things that are not objects. A variety of object implementations can be supported, including separate servers, libraries, a program per method, an encapsulated application, an object-oriented database. Through the use of additional object adapters, it is possible to support virtually any style of object implementation.

2.3 Object Adapter

Generally, object implementations do not depend on the ORB or how the client invokes the object. Object implementations may select interfaces to ORB-dependent services by the choice of Object Adapter. When an invocation occurs, the ORB Core together with the Object Adapter and The Skeleton arrange that a call is made to the appropriate method of the implementation.

An Object Adapter, a typical structure of which is shown in Figure 4, is the primary way that an object implementation accesses ORB services such as object reference generation. An object adapter exports a public interface to the object implementation, and a private interface to the skeleton.

An object adapter called the **Basic Object Adapter** (BOA) should be available in every ORB implementation. Although the BOA will generally have an ORB-dependent implementation, object implementations that use it should be able to run on any ORB that supports the required language mapping, assuming they have been installed appropriately.

CORBA specification mentions two other Object Adapters: **Library Object Adapter** (LOA) and **Object Database Adapter** (ODA). An LOA is primarily used for objects that have library implementations. It accesses persistent files but does not support activation or authentication because the objects are assumed to be in the clients program. An ODA uses a connection to a Database Management Systems (DBMS) to provide access to the objects stored in the database. Since a DBMS provides the methods and persistent storage, objects may be registered implicitly and no state is required in the object adapter.

3 OBJECT DATABASE ADAPTERS

An Object Database Adapter (ODA) allows object implementations to be written in the database programming language of the ODBMS, that is a language incorporating persistence into the programming environment. The object implementation is still responsible for managing the persistence state of the objects it implements but the task of an object implementor is much simpler in the programming environment provided by ODBMS. Besides persistency other database features like data consistency and crash recovery are available to the object implementation (Reverbel 1996a).

The importance of integrating ORB and ODBMS is widely recognised. Many vendors of ORBs and ODBMSes have announced plans to integrate their products. However, due to the commercial interests involved only a few design and implementation plans have been published: Sunrise ODA (Reverbel 1996a, 1996b, 1996c) and Orbix+ObjectStore Adapter (Iona 1995b).

3.1 The Sunrise ODA

The motivation behind the Sunrise ODA was the fact that object persistence was not sufficiently supported in CORBA-based environments. The initial design of the Sunrise ODA was for Iona's Orbix and Object Design's ObjectStore. Currently there are also releases for Orbix, mSQL, and Sunrise's own object-relational mediator as well as for VisiBroker and ObjectStore.

Rather than replacing the BOA, the Sunrise ODA is an add-on to the BOA (see Figure 5). The Sunrise ODA is implemented as a library that uses and extends the BOA services. It does not maintain any ORB-specific information in the database. Therefore, schema evolution is not needed when the Sunrise ODA is ported to another ORB. It is even possible that a single database is simultaneously accessed through CORBA servers based upon different ORBs.

3.2 Orbix+ObjectStore Adapter

IONA Technologies Ltd. has integrated its Orbix with the ObjectStore OODBMS. The result of this integration is called Orbix+ObjectStore Adapter (OOSA). The OOSA makes Orbix objects persistent by storing them in ObjectStore. On the other hand, ObjectStore objects can be accessed remotely by making them CORBA compliant. Due to commercial interests involved design and implementation information on OOSA is only partially public (Iona 1995a, 1995b).

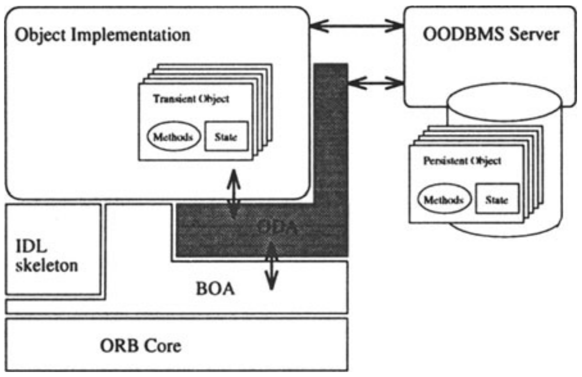

Figure 5 Sunrise ODA.

Orbix and ObjectStore are both object-oriented and based on C++ but their objectives are quite different. ObjectStore provides powerful support for persistence but it supports only a limited form of distribution. Orbix provides a flexible distribution model but has only basic support for persistence.

The integration allows an object to benefit both from persistency available in ObjectStore and from distribution transparency available in Orbix. The integrated system has three different types of processes (see also Figure 6):

1. **ObjectStore servers**: The innermost part in Figure 6 contains a set of ObjectStore servers. They are responsible for saving objects.
2. **Orbix servers**: The middle part contains a set of Orbix servers. These processes serve Orbix clients and are clients of ObjectStore servers.
3. **Orbix clients**: They reside in the outer part and make remote invocations on the objects provided by the Orbix server processes.

Naturally, a process can be both an Orbix client and an Orbix server when the process contains Orbix+ObjectStore objects and makes remote calls to other Orbix+ObjectStore objects. An object can also use ObjectStores distribution support. An object can be loaded into several Orbix server processes. In this case ObjectStore controls the concurrent access to the copies of the object.

Orbix clients do not necessarily be aware whether or not the called object is an object both in Orbix and in ObjectStore. References to both types of objects are transmitted in the same way. If the given reference points to an ObjectStore object, the object will be automatically loaded into the server process.

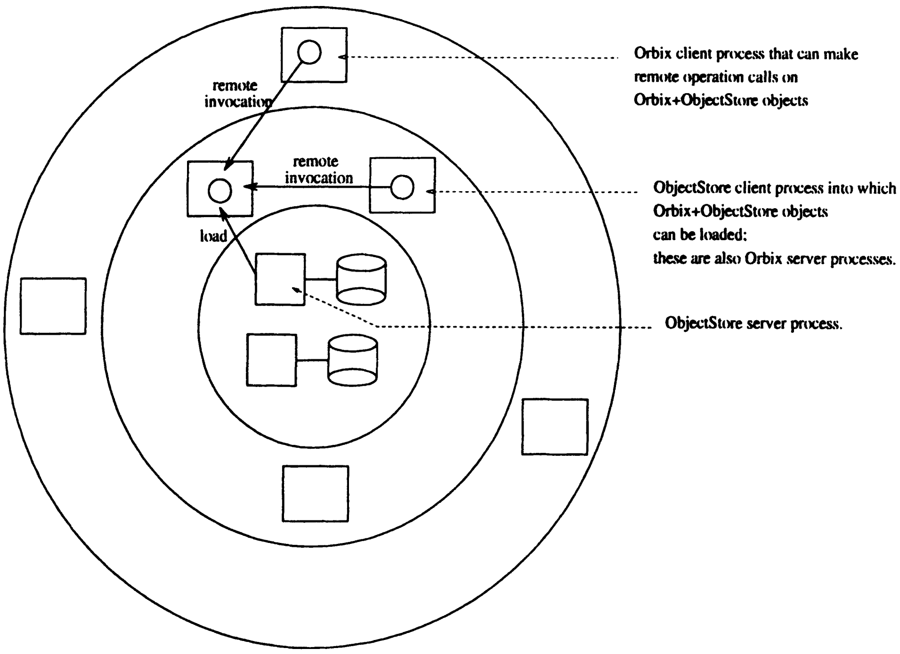
remote invocation
remote invocation
load
Orbix client process that can make remote operation calls on Orbix+ObjectStore objects
ObjectStore client process into which Orbix+ObjectStore objects can be loaded: these are also Orbix server processes.
ObjectStore server process.

Figure 6 Process types of OOSA.

4 RODAIN OBJECT-ORIENTED DATABASE ADAPTER

The **RODAIN Object-Oriented Database Adapter** (ROODA) is the way how we are going to provide CORBA clients an access to the RODAIN Database (Niklander *et al.* 1997). The ROODA will provide:

1. the OMG Persistent Object Service (POS),
2. the OMG Object Transaction Service (OTS),
3. an IDL interface to an ODMG-93 (Cattell 1994) compliant OQL interface as specified in the OMG Object Query Service (OQS), and
4. an flexible way of registering application specific IDLs to the database based on the OMG Dynamic Skeleton Interface (DSI).

4.1 ROODA Architecture

Figure 7 depicts how the CORBA access to the RODAIN Database will be organised in our prototype implementation. CORBA clients use ORB to obtain services available in our CORBA interface to the RODAIN Database. The ROODA acts as an interpreter between the ORBs and the RODAIN Database.

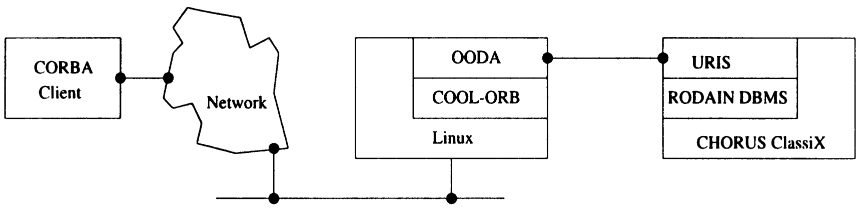

Figure 7 Interactions between CORBA clients, ROODA, and RODAIN database.

There are certain similarities between the OMG and ODMG-93 specifications that help us in building the interfaces:

- The data model used in RODAIN (Kiviniemi *et al.* 1996) is a real-time extension of the object model specified by ODMG-93 (Cattell 1994) which, in turn, is a superset of the object model specified by OMG (OMG 1993).
- The Object Definition Language (ODL) used in ODMG-93 is a superset of Interface Definition Language (IDL) of OMG.
- OMG recommends to use ODMG-93 as a standard interface for Persistent Object Service (POS).

In our prototype we have divided the Object Database Adapter into two parts. The ROODA, which interacts with the CORBA clients, runs on a front-end node. The ROODA use an internal interface of URIS to access the database system.

4.2 Object Mapping

At some level of abstraction there exists only objects in ODMG-93 and OMG. For example, the database is an object and the fetch operation is done by a method call to the database object. Changing the value of an attribute is done by a method call. Actually all of the functionality in the system is modelled by method calls to objects. Therefore, ORBs that access the database need to include information about object classes. However, the all the information should not be revealed outside the database. In particular, problems in distributed updates make it necessary to restrict the access from ORB to the class and type definitions (schema) in the database.

One of the primary advantages of using an OODA instead of POS is the fact that an object reference for every object in the database does not need to be registered with the ORB. Considering the case when the database consists of millions of objects this is a huge advantage. However, knowing only the logical name of the database is not sufficient for making queries. The ORB must have some information available about the contents of the database. Therefore, the ROODA maintains a copy of the database schema. As Figure 8 outlines, the classes defined in schema map CORBA-objects into database objects.

The ROODA also acts as the border between CORBA and RODAIN. There are certain information in RODAIN, for example the complete database schema,

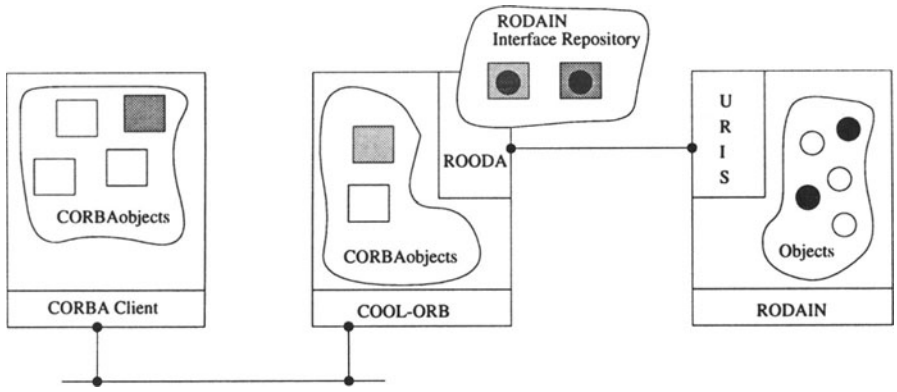

Figure 8 Mapping between CORBA and RODAIN objects.

which is not to be shown outside without strict control. Results of the queries from RODAIN are database objects but results given to the CORBA client are CORBA objects. If the result of a query leads to another query, the ROODA should know how the information from CORBA attaches itself to RODAIN objects. This means that there has to be some way of translating messages in ROODA. This translation is done by using an Interface Repository which includes the database schema that has interface definitions of all classes in the database. The definitions are expressed in OMG IDL. When we restrict the use of the database schema only to ROODA, not other parts of CORBA, we can be sure of the proper use of the schema.

4.3 Functionality

The functionality to be implemented in the ROODA can be semantically separated into two independent parts:

1. The ROODA provides **database access** to the RODAIN system. The CORBA interface is simply a way of using the database. In this role the ROODA provides only those tasks which directly use the database for retrieving, updating, and deleting objects in the database. This functionality is defined in ODMG-93 (Cattell 1994) and in the OMG Object Query Service.
2. The ROODA provides **database services** to CORBA clients. In this role the ROODA provides to CORBA clients the possibility of storing persistent objects in a way defined in POS. The ROODA can also take part in a distributed transaction as specified in OTS. The ROODA can also be the co-ordinator of a distributed transaction that only involves RODAIN Database nodes.

How different semantics these parts have, they use the same basic services of RODAIN. When a RODAIN user wants to change the database schema, schema management functions are to be used. When a CORBA client needs to store an object of a certain class for the first time, the class must be stored into database. In this task the ROODA uses the same schema management functions.

Below we examine the functionality provided by ROODA in terms of the relevant standards; which parts of different standards the ROODA must provide. The RODAIN Database system is based on the ODMG-93 standard. An OODA is introduced but not specified in CORBA references.

The **ODMG-93** standard describes the functionality relevant to use in a database. In order to allow a client fully to exploit the benefits of any database system, most aspects of the ODMG-93 standard must be supported. The RODAIN Data Model (Kiviniemi *et al.* 1996) is a real-time extension of the ODMG-93 object model. The functionality include means of data management: retrieval, modification, and deletion. Inherently, using a database through its own management system, other capabilities of the databases are also available: the so-called ACID properties, recovery, concurrency control, indexing.

An ODA is mentioned in the **CORBA reference** as an interface to databases. In addition, an OODA can be used to implement some of the functionalities described in CORBAservices. The ROODA will implement the functionality described in the Persistent Data Service module of POS (OMG 1996b). The ROODA will also also roles described in Object Transaction Service (OTS). In the OTS context the ROODA can be the co-ordinator of an internal RODAIN transaction or the recoverable server in a normal or nested transaction. In addition, parts of Object Query Service (OQS) are included into the ROODA through the OQL interface.

In order to become a production system in telecommunications, new services and service features must be easily introduced. A generic OQL-based queries with RODAIN Interface Repository service consumes too much resources. Therefore, service specific interfaces are needed. Currently we are examining whether or not the OMG Dynamic Skeleton Interface (DSI) can be used to as the means of deploying service-specific interfaces to the database.

Persistent Object Service

The Persistent Object Service (POS) provides a common interface to the mechanism used for retaining and managing the persistent state of objects. The POS has the primary responsibility for storing the persistent state of objects. Figure 9 shows the main components involved in the Persistent Object Service.

A *Client* is an application that uses a *Persistent Object* (PO). It is common for clients to need to control or to assist in managing persistence. In particular, the timing of when the persistent state is preserved or restored, and the identification of which persistent state is to be used for an object, are two aspects often of interest to clients. However, the client of the object can be completely ignorant of the persistence mechanism, if the object chooses to hide it. *Persistence Identifier*

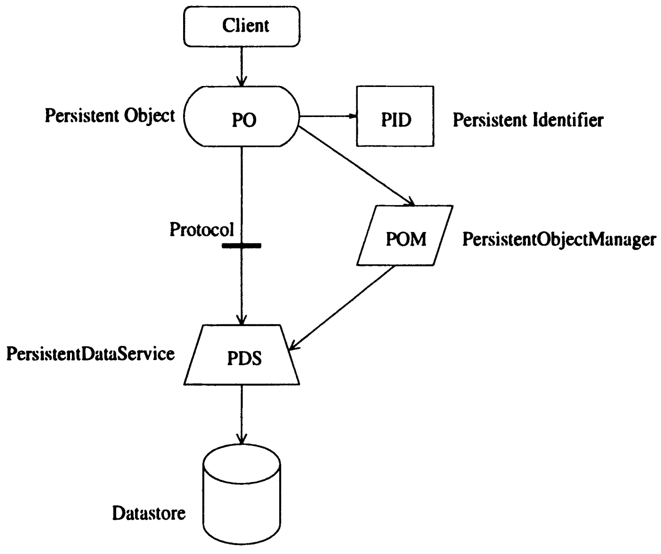

Figure 9 Major components of the POS and their interactions.

(PID) describes the location of persistent data of an object in some *Datastore* and generates a string identifier for that data.

The most crucial part of an *Object Implementation* is often in the definition and manipulation of persistence. The first decision the object makes is what interface to its data it needs. The POS captures that choice in the selection of the *Protocol* used by the object. The Protocol provides one of several ways to get data in and out of an object. The POS also defines a *Persistence Object Manager* (POM) that handles much of the complexity of establishing connections between objects and storage. The POM allows new components to be introduced without affecting the objects or their clients. An object has a single POM to which it routes its high-level persistence operations.

The *Datastore* provides one of several ways to store the data of an object independently of the address space containing the object. By having an interface that is hidden from objects and their clients, a Datastore can provide service to any and all objects that indirectly use the Datastore interface.

Persistent Data Service (PDS) actually implements the mechanism for making data persistent and manipulates it. The PDS provides a uniform interface for any combination of Datastore and Protocol. It co-ordinates the basic persistence operations for a single object. The PDS interacts with the object to get data in and out of the object using a *Protocol.* A protocol may consist of calls from the object to PDS, calls from PDS to the object, implicit operations implemented with hidden interfaces, or some combination. The Persistent Object Service specification defines three Protocols:

1. the *Direct Attribute Protocol* (DA protocol),

2. the *ODMG Protocol*, and
3. the *Dynamic Data Object Protocol* (DDO protocol).

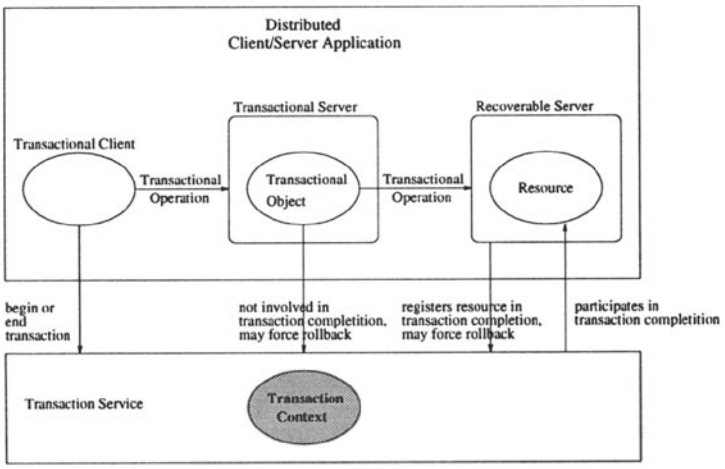

Figure 10 Basic elements of the Transaction Service [10-4/CORBAservices].

Object Transaction Service
The Object Transaction Service (OTS) provides transaction synchronisation across the elements of a distributed client/server application. A transaction can involve multiple objects performing multiple requests. The scope of a transaction is defined by a transaction context that is shared by the participating objects. The OTS places no constraints on the number of objects involved, the topology of the application or the way in which the application is distributed across a network.

Applications supported by the OTS consist of the entities shown in Figure 10:

- *Transactional Client* (TC) is an arbitrary program that can invoke operations of many transactional objects in a single transaction. The program that begins a transaction is called the transaction originator.
- *Transactional Object* (TO) is an object whose behaviour is affected by being invoked within the scope of a transaction. A transactional object typically contains or indirectly refers to persistent data that can be modified by requests. The term non-transactional object refers to an object none of whose operations are affected by being invoked within the scope of a transaction. Transactional objects are used to implement two types of application servers: *Transactional Server* and *Recoverable Server.*
- *Recoverable Objects* are objects whose data is affected by committing or rolling back a transaction. A recoverable object must participate in the Transaction Service protocols by registering an object called a resource with

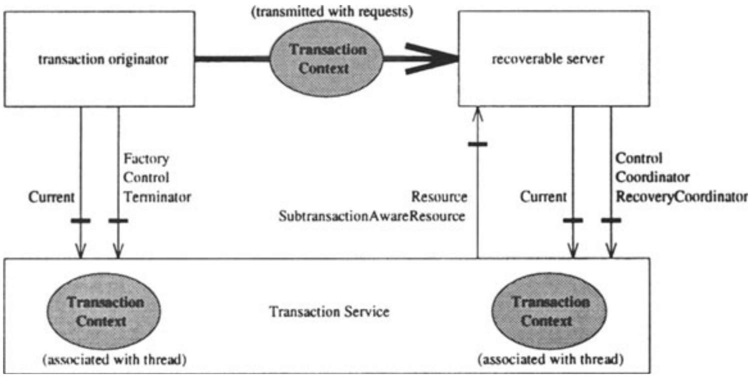

Figure 11 Interfaces of the Transaction Service [10-1/CORBAservices].

the Transaction Service. The Transaction Service drives the commit protocol by issuing requests to the resources registered for the transaction. A recoverable object typically involves itself in a transaction because it is required to retain in stable storage certain information at critical times in its processing. When a recoverable object restarts after a failure, it participates in a recovery protocol based on the contents (or lack of contents) of its stable storage.

- *Transactional Server* is a collection of one or more objects whose behaviour is affected by the transaction but these objects have no recoverable states of their own. A transactional server does not participate in the completion of the transaction but it can force the transaction to be rolled back.
- *Recoverable Server* is a collection of objects, at least one of which is recoverable. It participates in the protocols by registering one or more Resource objects with the Transaction Service. The Transaction Service drives the commit protocol by issuing requests to the resources registered for a transaction.

Figure 11 illustrates the major components and interfaces defined by the Object Transaction Service: **Transaction originator** is an arbitrary program that begins a transaction. **Factory** interface is used by the originator to create a transaction. **Control** interface allows an explicit management or propagation of the transaction context. **Terminator** interface is used by the transaction originator to commit or rollback the transaction. **Coordinator** interface is available to a recoverable server. **Resource** interface, which implements the two-phase commit protocol, is registered by the recoverable server to the Transaction Service. **SubtransactionAwareResource** interface can also be registered by the recoverable server to track the completion of subtransactions. **RecoveryCoordinator** interface can be used in certain failure cases to determine the outcome of transaction and to coordinate the recovery process with the Object

Transaction Service. **Current** interface defines operations that allow a client of the OTS to explicitly manage the association between threads and transactions. The interface also defines operations that simplify the use of the OTS.

Object Query Service

The Object Query Service (OQS) (OMG 1996b) provides selection, insertion, updating, and deletion on collections of objects. These operations are defined as predicate-based queries. Operations are executed to *source* collection and they may return *result* collections of objects. The result collections of objects may be either selected from source collections or produced by query evaluators. The source and result collections may be typed.

The specification of OQS is based on the following design principles:

- The OQS should allow arbitrary user objects to invoke queries on arbitrary collections of other objects. Such queries may specify values of attributes, invoke operations, and invoke arbitrary OMG Object Services.
- The OQS should support the OMG architecture. Therefore, it should allow querying against any objects, with arbitrary attributes and operations.
- The OQS must allow use of performance enhancing mechanisms such as indexing.
- The OQS should smoothly and efficiently co-operate with the internal mechanisms of database systems, especially in specifying collections and in using indexing.
- The OQS must also allow the native systems to contribute in specifying collections and indexing.

The specification of OQS is based on existing standards for query. When necessary to accommodate other design principles the model is extended. In OQS the *query evaluator* can be *nested* and *federated* like the transactions in the Object Transaction Service. Objects may participate in the Query Service in two ways:

- **Any CORBA object is queryable**. The Query Evaluator evaluates the query predicate and query operations. Query operations are performed by invoking operations on that object through its published OMG IDL interface. If an operation is not supported, then an exception is triggered. This mechanism provides generality but prevents optimisation.
- **Objects may participate as members of collections**. The collection has a specific query interface. In other words the collection is a Query Evaluator. This way allows Query Evaluators or any associated native query system to evaluate the query using the internal optimisation.

Table 1 OQS modules and their interfaces

CosQueryCollection

Interface	*Purpose*
CollectionFactory	To create collections
Collection	To represent generic collections
Iterator	To iterate over collections

CosQuery

Interface	*Purpose*
QueryLanguageType	To represent query language types
QueryEvaluator	To evaluate query predicates and execute operations
QueryableCollection	To represent the scope and result of queries
QueryManager	To create query objects and perform query processing
Query	To represent queries

The Query Service provides definitions and interfaces for creating and manipulating collections of objects. The collections are defined as objects with methods for inserting and deleting members. Associated iterators are defined in order to allow manipulation of collections. The members of the same collection may be of different types. The CORBA collections may directly map to collections managed by native query systems. These native collections may include arbitrary CORBA objects. The Query Service is independent of any specific query language. A Query Service provider must support either SQL or OQL-93 query language, that is the Object Query Language defined in the ODMG-93 standard.

The Query Service defines two types of service; see also Table 1:

- Collections include two interfaces to create and manipulate collections of objects. The *Collection* interface includes operations for creating and manipulating collections. The *Iterator* interface defines operations for traversing over and retrieving objects.
- The *Query Framework* interface defines a framework for an object query. The *QueryLanguageType* interface classifies query language types defined in OMG IDL. The *QueryEvaluator* interface defines basic operations for query evaluation. The *QueryableCollection* interface defines the result of the query. The *QueryManager* defines a more powerful QueryEvaluator which can create arbitrary Query objects. Query objects can provide graphical query construction, pre-compilation and optimisation of a query, asynchronous query execution.

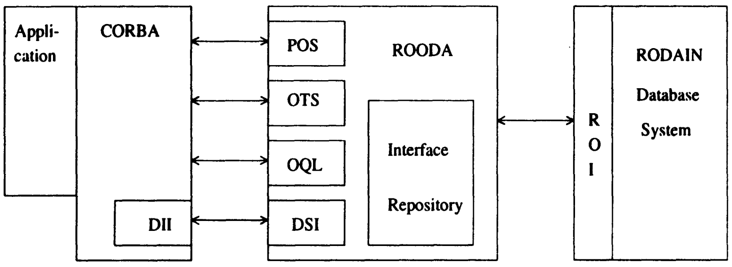

Figure 12 OODA interfaces to CORBA and RODAIN.

4.4 Interfaces to ORB

Since the services provided by any ODA must be transparent, the ORB interfaces must be constructed using existing service interfaces. The interfaces that an ODA should, at least partially, support include POS, OTS, OQS, and DSI. Figure 12 depicts the interfaces to those services the functionality of which the ROODA is designed to support.

The interfaces are:

- In the POS context the ROODA must provide functionality described in Persistent Data Service module. ROODA should interact with the RODAIN Database in order to get data in and out. The ROODA should also interact directly with the object itself in order to get data in and out using protocol. OMG endorses ODMG-93 as a standard interface for storing persistent object state.
- In the OTS context the ROODA must provide functionality described for Recoverable Server. The ROODA must have properties or understand messages of/from objects *Current*, *Control*, *Coordinator*, *RecoveryCoordinator*, *Resource*, and *SubTransactionAwareResource*.
- In the OQS context the ROODA must act as a *Query Evaluator*. Query Evaluator is responsible for evaluating the query predicate and performing all query operations by invoking operations on that object.
- In the DSI context the ROODA offers to a CORBA client a way of dynamic invocation of objects. On the client side the counterpart of DSI is the *Dynamic Invocation Interface* (DII).

4.5 Interface to RODAIN

The interace to RODAIN URIS is built on the ODMG-93 standard data model, which is used in RODAIN. The functions described in the ODMG-93 standard (Cattell 1994) or in our real-time extensions (Kiviniemi *et al.* 1996) offer a low-level interface into RODAIN. The basic idea of this object oriented approach is that RODAIN contains objects for which the CORBA client can make method calls. Or the client can fetch the objects to its own memory, modify them and finally return them into the database. Due to the nature of ODMG-93 data model even the fetches are method calls which are applied to the database-object.

The ODMG-93 model's interface may be too low-level for sophisticated use. Therefore, we have will use an intermediate language between the ROODA and the RODAIN Database based on clauses in relational algebra. Other URISes that interpret other protocols also use algebraic clauses. The intermediate language based on such clauses can, in a rather straightforward manner, be optimised by a single optimiser regardless the used protocol.

5 CONCLUSIONS

Object technology based on the OMG specifications will be the next step in telecommunications software. As introduced in this paper CORBAservices standardised by the OMG provide a sound basis for database access in telecommunications. The Persistent Object Service and Object Transaction Service are the ways in which CORBA objects can be stored and manipulated in a database system. The Object Query Service and Dynamic Skeleton Interface provide the other side of the coin. When CORBA is used as an enabling technology in telecommunications, the databases containing the operational, administrative, and management data must be accessed through CORBA.

An Object Database Adapter is the key component when objects and services available in telecommunications databases are provided to CORBA clients. A typical database includes millions or billions of objects. If all these objects must be registered as CORBA objects, any ORB implementation will not scale to the needs of telecommunications. Therefore, the ODA must provide an interface repository that dynamically provides query results as CORBA objects. Another ability needed in the ODA is dynamic registration of application specific interfaces for database services.

6 REFERENCES

Barr, W.J., Boyd, T. and Inoue, Y. (1993) The TINA initiative. *IEEE Communications Magazine*, **31**, 3, 70-6.

Cattell, R.G.G. (ed.) (1994) *The Object Database Standard: ODMG-93*. Morgan Kauffmann, San Francisco, Calif.

Iona (1995a) *The Orbix Architecture*. Iona Technologies Ltd., Dublin.

Iona (1995b) *The Orbix+ObjectStore Adapter*. Iona Technologies Ltd., Dublin.

ITU-T Recommendation M.3010 (1992) *Principles for a Telecommunications Management Network*. International Telecommunications Union, Geneva.

ITU-T Recommendation Q.1201 (1993) *Principles of Intelligent Network*. International Telecommunications Union, Geneva.

Kiviniemi, J. and Raatikainen, K.E.E. (1996) *Object Oriented Data Model for Telecommunications*. Report C-1996-75, University of Helsinki, Department of Computer Science, Finland.

Niklander, T., Kiviniemi, J. and Raatikainen, K.E.E. (1997) A real-time database for future telecommunication services, in *Proceedings of 2IN'97*, Chapman & Hall, London.

OMG (1993) *Object Management Architecture Guide*. John Wiley & Sons, New York.

OMG (1995) *CORBAfacilities: Common Facility Architecture*. John Wiley & Sons, New York.

OMG (1996a) *CORBA: Common Object Request Broker Architecture and Specification*. John Wiley & Sons, New York.

OMG (1996b) *CORBAservices: Common Object Service Specification*. John Wiley & Sons, New York.

Reverbel, F. (1996a) ORB/ODBMS integration. URL="http://www.acl.lanl.gov/~reverbel/orb_odbms.html".

Reverbel, F. (1996b) ORB/ODBMS integration in the Sunrise project. URL= "http://www.acl.lanl.gov/~reverbel/reverbel_orb_odbms.html".

Reverbel, F. (1996c) *Persistence in Distributed Object Systems: ORB/ODBMS Integration*. Ph.D. Dissertation, University of New Mexico, Computer Science Department.

7 BIOGRAPHY

Pasi Porkka received the B.Sc. degree in computer science from the University of Helsinki, in 1996. He is currently a research assistant in RODAIN research project and completing his M.Sc. thesis about integrating CORBA to object-oriented database. His research interests include real-time databases, CORBA architecture and data mining from relational databases.

Kimmo Raatikainen received the Ph.D. degree in computer science from the University of Helsinki, in 1990. He is currently an associate professor in computer science at the University of Helsinki. He is a member of ACM, IEEE (Communications and Computer Societies), and IFIP TC6 Special Interest Group of Intelligent Networks. His research interests include nomadic computing, telecommunications software architectures, and real-time databases.