Introduction

As Birrel *et al.* pointed out in [3], the object-oriented programming methodology applies beautifully to distributed computing. Since object-oriented programming enforces encapsulation, clients can only access the state of an object via the object's methods. The method calls are therefore a natural place to insert the communication needed in a distributed environment. Even some words of the object-oriented terminology ("message", "receiver") reveal the potential affinity between object-oriented programming and distributed computing. At the root of this affinity is the concept of separation, fundamental to both disciplines: in one of them, it appears as the logical separation among encapsulated objects; in the other, as the physical separation among distributed entities.

Although research systems based on this observation started to appear by the mid 1980s, distributed objects have not yet attained mainstream status. More than the many technical issues involved in the design of a distributed object environment[1], the lack of a standard for such environments prevented their widespread use. This situation is now changing with the emergence of two competing standards: CORBA, the Object Request Broker (ORB) architecture specified by the Object Management Group (OMG), and COM, the Component Object Model upon which Microsoft based its Object Linking and Embedding (OLE) infrastructure.

Nearly all major software and hardware vendors are backing the OMG architecture, Microsoft being the single and notable exception. Due to Microsoft's position in the software industry, however, it is likely that COM will become an alternative standard — a *de facto* standard for personal computer platforms[2]. CORBA prevalence is

[1] See the overview section of [3] for a brief discussion of some of these issues.

[2] COM is Microsoft's proprietary solution, not a formal standard. Only its non distributed (single-

expected on all other platforms, and especially on POSIX systems. It appears that both standards are here to stay, and will be coexisting in the foreseeable future. The present work addresses CORBA [37], because:

1. several ORB implementations are available now;

2. as the *de jure* standard, CORBA is an open and cross-platform solution;

3. most neutral object technology experts agree that the OMG architecture is technically superior to OLE/COM.

Our Subject

In the OMG model for distributed object systems [40], the ORB is the software component through which objects transparently make requests and receive responses. It provides interoperability between applications written in different languages and running on different machine architectures, seamlessly interconnecting multiple object systems in heterogeneous distributed environments.

In many situations, objects should outlive the processes that implement them. These situations include most practical scenarios in which distributed objects are desirable: in real-world applications, persistence is almost always a requirement. While CORBA does not support object persistence directly, its open architecture includes an *object adapter*, an intermediary between an object implementation and the ORB core. Object persistence can then provided by the object adapter.

The CORBA specification defines a Basic Object Adapter (BOA) that does not provide object persistence. As an example of another object adapter that might be useful, CORBA briefly mentions an adapter that makes objects stored in an object-oriented database accessible through the ORB. This idea is pursued in the Appendix B of the ODMG standard [7], which identifies a number of issues involved in using an Object Database Management System (ODBMS) in a CORBA environment and proposes an Object Database Adapter (ODA) to realize the integration of the ORB with the ODBMS.

---

machine) version is currently available. Network COM will not be shipping before late 1996, with Cairo's Network OLE.

An Object Database Adapter allows object implementations to be written in the database programming language of the ODBMS, a language that incorporates persistence into the programming environment. The object implementation — a CORBA server — is still responsible for managing the persistent state of the objects it implements, but the object implementor's task is much simpler in the programming environment provided by ODBMS. Besides persistence, other database features — data consistency in the presence of concurrent accesses, crash recovery, and so forth — are available to the object implementation.

In existing database systems, regardless of their data model (object-oriented or relational), database clients must have some knowledge of the database schema. In the case of an object-oriented DBMS, clients need to know the object layout. In a relational DBMS, views can be used for data independence. But relational clients still need to know the external (view level) schema. By contrast, ORB/ODBMS integration makes database objects accessible to CORBA clients without exposing the database schema to these clients. The data members and the layout of a persistent CORBA object remains private, only its interface (a set of methods) is made public. This is specially interesting for web browser access to databases. With the integration of the Java language ([11], [55]) into the ORB environment, Java applets can interact with persistent CORBA objects through domain-specific interfaces, without any knowledge of how the objects are actually stored.

ORB/ODBMS integration leads to ORB-connected multidatabases. Together with OTS, the Object Transaction Service[3] specified by the OMG in [39], it enables the construction of distributed object databases that are truly heterogeneous, even with respect to the DBMS software running on the various database server nodes.

Storing CORBA Objects in an Object Database: The Issues

The gap between object access times in the ORB and in the ODBMS environment is the first issue an Object Database Adapter must address. Because CORBA clients access objects via remote method invocation, access times for CORBA objects are expressed in milliseconds. Because ODBMSs keep an object cache at the client side,

[3]OTS implements the two-phase commit protocol [12] in a CORBA environment. Its model can be viewed as an object-oriented extension of the X/Open DTP model [60].

ODBMS clients can access objects much faster: access times for ODBMS objects are typically expressed in microseconds.

Due to the this performance gap, an Object Database Adapter cannot force all accesses to the objects in a database to be made through the ORB remote method invocation mechanism. In the case of a large collection of very small objects, the overhead would be unacceptable. Instead, the ODA should let the object implementor choose a suitable subset of database objects, presumably the higher level ones, to be accessed as CORBA objects. Since this subset may still be large, individual registration of its objects with the ORB is not practical. The ODA should allow a subset of database objects to be accessed through the ORB, without requiring an explicit registration call for each object.

Moreover, in the common case of an ODBMS that adds database features to C++, an object implementation cannot simply store in an object database the CORBA objects it implements:

- It would be a waste of database space: a C++ CORBA object has ORB-specific data members that should not be stored. It typically also has a pair of hidden `vbase` and `vtable` pointers for each interface class in its inheritance chain up to `CORBA::Object`.

- More importantly, the CORBA operations `duplicate` and `release` update the object's reference count. If the reference count were actually stored in the database, every operation on the object would have to be performed within an update transaction, because `duplicate` and `release` appear everywhere.

- Yet more importantly, ORB implementations keep a per-process table of active objects: a new entry is inserted into this table whenever the constructor of a CORBA object is invoked by the corresponding process. In a C++-based ODBMS, however, the constructor of a persistent object is only invoked when the object is added to the database. As far as the ORB is concerned, CORBA objects stored by other processes (including previous runs of the same program) would not be active.

An Object Database Adapter has to solve these problems, ideally in a way that makes persistent CORBA objects appear exactly like ordinary database objects. As much as possible, object implementors should be unaware that a persistent CORBA object does not really live in the database.

Persistence of CORBA Object References

Besides providing persistence to CORBA objects, the Object Database Adapter must also provide persistence to the corresponding object references. In CORBA, *persistence of object references* means that "a client that has an object reference can use it at any time without warning, even if the (object) implementation has been deactivated or the (server) system has been restarted" [37].

With persistence of object references, it makes perfect sense for a client to store an object reference for later use. References to persistent CORBA objects implemented by server X can be stored by server Y (a client of server X) and vice-versa, thereby enabling the construction of ORB-connected multidatabases. In such a multidatabase, references to remote objects are used to express relationships between CORBA objects implemented by different servers. If distributed transactions are needed, they can be supported by a TP monitor that implements the Object Transaction Service.

Storability of CORBA Object References

Because an object reference is opaque and ORB-dependent, CORBA provides operations that convert an object reference to string and vice-versa. Object references are stored as strings; upon retrieval they must be converted back to their native form.

Translation to and from string format provides maximum flexibility, allowing object references to be kept in any media. In an ODBMS environment, however, object storage and retrieval are transparent to the programmer. The need for explicit translation of stored references does not prevent the construction of ORB-connected multidatabases, but is unnatural in the context of an object database: ODBMS users expect stored references that behave like any other database object.

Transparent storability of CORBA object references is yet another desirable feature of an Object Database Adapter. It eliminates the need for explicit object reference conversions, both before storage and after retrieval, and allows transparent use of stored references to invoke methods on possibly remote objects.

Scope of and Organization of this Dissertation

According to Cattell [6], "more work in both standards and implementation is needed to realize the integration of ODBMSs and OMG ORBs". In this dissertation we present:

- A discussion of the design issues involved in integrating ORBs and ODBMSs.

  Our experience designing and implementing Object Database Adapters is conveyed here. We have implemented two approaches to persistence of CORBA objects. The first one, which we call *pseudopersistence*, makes object references persistent, but not transparently storable. The second approach, which we call *virtual persistence*, provides transparent storability to object references, both in the case of references to local objects and in the case of references to remote objects. These approaches, as well as a smart pointer-based realization of transparently stored references, are discussed in considerable detail. We stress the ORB and ODBMS features each approach depends upon, and point out the ones not fully standardized at the present time.

- A report on the utilization of an Object Database Adapter in a real-world distributed application.

  We discuss ODA usage in TeleMed, a distributed, object-oriented, CORBA-based telemedicine system developed at the Los Alamos National Laboratory.

- A performance evaluation of ORB/ODBMS integration approaches.

  Our quantitative results reveal the good performance of pseudopersistence, effectiveness of object caching, the promising performance of smart pointer-based realizations of transparently stored references, and the penalty paid for using virtual persistence with the single commercially available ODBMS to which this approach is applicable.

The remainder of this dissertation is organized as follows: Chapters 1 and 2 review background material on ORBs and ODBMSs. Chapter 3 discusses the reasons for integrating an ORB with an ODBMS, the role of an Object Database Adapter, and ODA design issues. Chapter 4 is a report on our ODA implementation and its utilization in the TeleMed project. Chapter 5 presents a performance evaluation of different approaches to the integration of an ORB with an ODBMS. Finally, Chapter 6 contains our concluding observations and summarizes the contributions of this work.