Towards Reliable CORBA Integration vs. Service Approach

Pascal Felber, Benoît Garbinato, and Rachid Guerraoui

Laboratoire de Systèmes d'Exploitation, Département d'Informatique
École Polytechnique Fédérale de Lausanne, Switzerland

1 Introduction

Increasing need and interest for distributed technologies have led to the emergence of several object-oriented distributed architectures. One of the most important of these middlewares is the *Common Object Request Broker Architecture (CORBA)*, specified by the *Object Management Group (OMG)* [OMG95a]. It provides an object-oriented infrastructure that allows objects to communicate, independent of the specific platforms and techniques used to implement these objects. CORBA defines *objects* as the unit of distribution, each object providing a set of operations through a single *interface*.

CORBA provides the basic mechanisms for remote invocation through the *Object Request Broker (ORB)*, as well as a set of services for object management, e.g. Persistence Service, Transaction Service, etc [OMG95b]. Nevertheless, neither the ORB nor the existing services provide tools for building reliable and highly available applications. This can be considered as a major drawback for many of today's industrial applications, which require a high degree of reliability and availability. Crashes or network failures are some of the problems that frequently occur and which the programmer has to deal with.

Group communication is considered a powerful paradigm to support fault-tolerance and high availability through replication. Several systems have successfully experimented with the support of reliable and highly available distributed applications through group communication, some of which have an object-oriented architecture. Among these systems are Horus [RBC93], Isis [BCJ+90], Phoenix [Mal96], Totem [AMMS+95], and Transis [ADKM92].

The goal of this position paper is to discuss how to extend a CORBA environment with the groups. We present the two main approaches for adding group communication to CORBA: (1) the *integration* approach and (2) the *service* approach. The first approach consists of modifying and extending the Object Request Broker (ORB) with group communication. This approach has been adopted in both Orbix+Isis [II94] and Electra [Maf95]. The second approach consists of providing group communication as a service on top of the ORB. We compare the two approaches and we argue that the *service* approach is a better solution, as it follows the CORBA philosophy by preserving the modularity of the architecture [FGG96].

2 Background

Object-orientation and group communication technologies have been two distinct research domains until the last few years. Integrations of these technologies is a new and challenging task. This section recalls some background on both technologies.

2.1 Group communication

The group communication paradigm allows the provision of high availability through *replication* in a straightforward way (by gathering a set of replicas into a single group). We describe below the general group model as adopted by most group-oriented systems (i.e. Horus, Isis, Phoenix, Totem, Transis). The key mechanisms underlying group communication are *group multicasts* and *dynamic group membership*. A well known example of a group multicast primitive is the *total order multicast*, which ensures that all requests sent to a group will be received by all the members of the group in the same order. Dynamic group membership allows to change the composition of the group at run-time.

Since the members of a group usually share a common state, a new member has to receive the current state from the group when joining it. This is performed by a *state transfer* mechanism that transparently transmits the current state from an "old" member of the group to the new one. Group members must thus provide operations for "getting" and "setting" their shared state. Group members are also notified whenever a member joins or leaves the group — possibly because of a crash — so that any member knows the current composition of the group. This mechanism is called a view *change*.

When communicating with groups, clients do not need to know the number of group members, nor how the members divide and perform their work. A group of replicas can thus be viewed as an abstract fault-tolerant entity[1].

2.2 The CORBA model

Classical objects only live in a single program and the outside world does not know anything of them. In contrast, CORBA objects can live anywhere on a network. They can be accessed by local or remote clients via operation invocations. The basic mechanisms for remote object invocation are provided by the Object Request Broker (ORB), which acts as a ``software bus''. The language used to create server objects, and the system they are running on, are totally transparent to clients; the clients only need to know the interface the server objects publish, which acts as a ``binding contract'' between clients and servers. Interfaces are specified in an Interface Definition Language (IDL), which is implementation independent.

1. Transactions can also be used for fault-tolerant computing, but they do not provide high-availability.

Clients use *object references* to identify remote objects and to invoke operations on them. References are valid in the whole system and can thus be passed from one node to another. Furthermore, objects are not tied to a client or server role; they can act in turn as client and as server.

The basic communication platform provided by CORBA, that is the ORB, only supports minimal mechanisms for remote inter-object invocation. More specific functionalities (e.g. persistence) are added outside of the ORB as *CORBA services*, standardized by the OMG. A service is basically a set of CORBA objects with their corresponding IDL interfaces, which can be invoked through the ORB by any CORBA client. The main characteristics of the objects composing a service is that they are not related to any specific application but are rather basic building blocks, usually provided by CORBA environments.

3 Adding group communication to CORBA

As pointed out in the previous section, several mechanisms that augment the basic functionalities of CORBA have been designed and adopted by the OMG as part of CORBA specification. Nevertheless, no support is provided for group communication[1]. This can be considered as a drawback for applications that require a high degree of availability and need to handle replicated objects.

We will now discuss the two main approaches for adding “object” group communication to a CORBA environment. The first one consists of integrating the notion of group within the ORB core, allowing a client to perform invocations to a group of objects as if it was a singleton object. The ORB performs the invocation to the replicated objects, gathers the replies, and returns them to the client. This approach provides transparency but requires a modification of the ORB core. The second approach consists of providing an external service that offers the basic primitives for managing groups of objects; the ORB knows nothing about groups. This approach is similar to the one adopted by the OMG to enhance CORBA with persistence, transactions, event channels, etc. Achieving complete group transparency is not straightforward with this approach, but the ORB is not modified and the modularity of the architecture is preserved.

3.1 Integration approach

The key idea of integration is that group communication is supported by the ORB core itself. We know of two ORBs which have adopted this approach, namely Orbix+Isis and Electra.

1. More precisely, there is no CORBA specification for group communication, though as we will point out in Section 3.1, some ORBs have integrated group communications in an ad hoc manner.

In this model, groups contain objects of the same type[1]. An object reference of type *T* is mapped to a group of objects of type *T*. The ORB is modified to distinguish references to singleton objects from references to groups of objects. Figure 3-1 (a) illustrates how a request is performed to a group of objects using *Orbix+Isis*. The request is given to the ORB, which recognizes that it is addressed to a group of objects and translates it into a call to an Isis multicast primitive. At the server sites, the relevant operation of each member is invoked by the ORB, which returns the result to Isis. Finally, replies are gathered at the client site, and returned to the client.

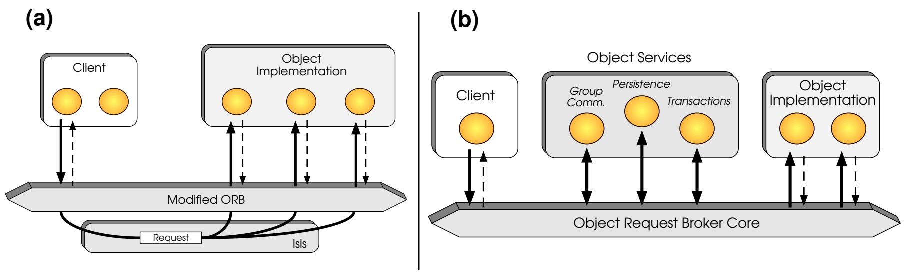

Fig. 3-1 The Orbix+Isis Model and the Service approach

By default, only one of the replies is returned to the client in order to keep the group invocation transparent. However the client is also able to access all the replies, if it is aware of the group, and wants to take advantage of it. The basic idea is to generate two types of functions from IDL definitions: (1) standard functions that conform to the language mapping and (2) special functions with sequences of values for *out* and *inout* parameters. The client uses the function with the signature that corresponds to its needs. If the client is not aware of groups, only the standard functions will be used. Electra [Maf95] and Orbix+Isis [II94] implement different variations of the integration approach.

- **Electra** extends the *Basic Object Adapter (BOA)*, adding the ability to *create*, *destroy*, *join* and *leave* a group, the notification of *view changes*, and the support for *state transfer*. Since the modification is performed at the BOA level, every CORBA object of the system can potentially be member of a group. All communication is performed using a toolkit that supports group communication (Isis or Horus). Electra has a modular architecture in the sense that it can use different underlying group communication systems. This model is “orthogonal” in the sense that all server objects are handled in the same way. However, it adds unnecessary complexity to objects that are not member of a group, and which only need BOA support.
- With **Orbix+Isis**, objects that can be group members have specific properties. Point-to-point communication goes through Orbix while multicasts are handled

1. The type of a CORBA object is defined by its IDL interface.

by Isis. To be a member of a group, an object must inherit from an abstract base class that extends the object with the notification of *view changes* and the support for *state transfer*. To handle object replication, the programmer can choose between two styles of object groups: *Active Replica* and *Event Stream*. The role of the server is fixed at compile time.

Another variation of the integration approach consists of providing a second object adapter in addition to the *Basic Object Adapter*, which is allowed by the CORBA specification. We would then have the basic adapter for standard objects and a dedicated adapter for member objects.

3.2 Service approach

The basic idea of the service approach is to provide group communication as a service on top of the ORB, and not as part of the ORB itself. The service ``plugs'' on the object bus, enhancing it with new functionalities. As shown in Figure 3-1 (b), the *Group Communication Service (GCS)* appears as a new CORBA service, besides other services (e.g. transaction service and persistence service), and also besides any other CORBA object.

The graphical representation of the service in Figure 3-1 (b) is completely independent of the effective location of the service. A service is viewed as a set of IDL interfaces, and it can be composed of several objects located at different nodes on the network. The service is inherently accessible from anywhere on the bus.

Using a service makes groups explicit; one has to invoke the Group Communication Service to join groups, multicast messages, etc. For a client to send a request to a group, the client must invoke a specific *multicast* operation on an object of the service. It is the latter which is then in charge of multicasting the request (with the desired semantics, e.g. total order) and collecting the replies. However, groups can be hidden at the client side using smart proxies, which can be automatically generated from IDL definitions. On the server side, member objects are required to support a specific set of operations to enable the service to *call back* to them to invoke some operation (view changes, state transfer, etc.).

3.3 Discussion

The main advantage of the integration approach versus the service approach is transparency. An object group is not distinguishable by a client from a singleton object that implements the same interface. Thus, a singleton object can easily be replaced by an object group to achieve high availability by simply recompiling its clients, as long as the latter exhibit a standard behavior (e.g. they do not need to access the complete set of replies resulting from an invocation to a group). This approach makes it possible to reuse existing clients with very few changes, and to let them access highly available components. In oder to achieve a similar level of transparency in the service approach, the usage of smart proxies is necessary. Any of the two approaches require server objects (i.e. group members) to deal with replication.

Another advantage of the intergration approach is that an implementation can be very efficient since group communication is performed directly by the ORB. Multicasting a request does not involve an intermediary object as in the service approach.

The cost of transparency is that the semantics of CORBA object references are modified. The *Object Model* of the CORBA specification defines an object reference as *an object name that reliably denotes a particular object. An object reference identifies the same object each time the reference is used in a request, and an object may be denoted by multiple, distinct references* [OMG95a]. With the integration approach, an object reference does not uniquely identify a single CORBA object on the network, but (possibly) a group of objects. Furthermore, the objects referenced can change over time (the reference can even designate an empty group).

The other drawbacks of the integration approach are the loss of portability (implementation is strongly ORB-dependent) and interoperability (both clients and servers have to use the same ORB implementation, which relies on proprietary group communication mechanisms[1].

The advantages of the service approach over the integration approach mainly concerns portability and interoperability. A service is not ORB-dependent (it is defined only in terms of an IDL interface), and by using the ORB communication primitives it benefits from the ORB's interoperability properties. It requires no modification of the ORB and is easily portable from one ORB to another.

A service offers a more powerful and flexible model since it makes the use of groups explicit, and therefore does not rely on a set of predefined behaviors (e.g. *Active Replica* execution style). It also makes it possible to group objects that support different IDL interfaces. Finally, multiple services can cooperate in the system, e.g. the *GCS* can use a Recovery Service to recover server objects from crash failures, or the Persistent Object Service to manage persistent replicas.

We believe that the service approach is more appropriate to a CORBA environment as it follows the OMG ``philosophy'': that is, the ORB should provide a minimal mechanism to break the boundaries of languages and address spaces, allowing heterogeneous distributed objects to interoperate. All additional functionality should be provided as services, i.e. specific CORBA objects with IDL interfaces. For instance, concurrency or persistence are needed by many applications, and some programming languages address these issues, but the OMG did not include any support for them in the ORB core. They rather defined external services, keeping the ORB core as simple as possible with the necessary tools common to all CORBA applications. Finally, a service is much more likely to be standardized than an ORB extension (or even a new object adapter).

---

1. Interoperability is generally achieved by using normalized protocols like the Internet Inter-ORB Protocol (IIOP) specified in the CORBA 2.0 specification. It is possible to provide an IIOP gateway for an ORB that uses a group communication subsystem, but the gateway represents a single point of failure.

4 Concluding remarks

The addition of group communication support to CORBA yields an environment that promotes reliability and high availability, as well as interoperability and portability.

In this paper we have discussed the problem of adding group communication to CORBA, focusing on two main approaches: the integration approach which consists of integrating group communications within the Object Request Broker (ORB) core, and the service approach which provides group communication as a service besides the ORB. We have pointed out the advantages and drawbacks of each approach, and we have expressed our preference for the service approach. This approach follows the CORBA philosophy by promoting interoperability and portability. It follows the design of the other functionalities that have been added to CORBA through services, such as persistence and transactions, which were specified in the OMG's Interface Description Language (IDL) and adopted as CORBA standards. Furthermore, in the specification of the CORBA services, the OMG mentions a future *Replication Service* intended to provide explicit replication of objects and consistency of replicated copies [OMG95b]. A Group Communication Service constitutes a fundamental building block to support replication.

References

1. [ADKM92] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a communication sub-system for high availability. In *Proceedings of the 22nd International Symposium on Fault Tolerant Computing Systems*, 1992.
2. [AMMS+95] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P.Ciarfella. The totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4):311--342, November 1995.
3. [BCJ+90] K. Birman, R. Cooper, T. A. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck, and M. Wood. The Isis System Manual. Dept of Computer Science, Cornell University, September 1990.
4. [FGG96] P. Felber, B. Garbinato, and R. Guerraoui. The design of a corba group communication service. In *Proceedings of the 15th Symposium on Reliable Distributed Systems*, 1996.
5. [II94] IONA and Isis. *An Introduction to Orbix+Isis*. IONA Technologies Ltd. and Isis Distributed Systems, Inc., 1994.
6. [Maf95] S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming*. PhD thesis, University of Zurich (Switzerland), February 1995.
7. [Mal96] C. Malloth. *Conception and Implementation of a Toolkit for Building Fault-Tolerant Distributed Applications in Large Scale Networks*. PhD thesis, Federal Institute of Technology, Lausanne (EPFL), 1996.
8. [OMG95a] OMG. *The Common Object Request Broker: Architecture and Specification*. OMG, 1995.
9. [OMG95b] OMG. *CORBAservices: Common Object Services Specification*. OMG, 1995.
10. [RBC93] R. Van Renesse, K. Birman, and R. Cooper. The HORUS system. Technical report, University of Cornell (NY), 1993.