**Abstract**

This paper presents the design of **OPEN EDEN**, an implementation of the Fault Tolerant CORBA specification based on the use of a group communication system, called **EDEN**. The design of **OPEN EDEN** has been driven by the desire to use only portable techniques (mainly portable interceptors) to implement the interaction between the **EDEN** group communication system and CORBA. The **EDEN** system itself is built by configuring, assembling and deploying components connected to a generic agreement component that implements a consensus algorithm. The resulted platform is particularly portable, since it can be easily migrated to different architectures, and transparent for applications programmers, since requests to a replicated service are transparently intercepted and redirected to the **EDEN** system. We discuss the main difficulties we encountered and we draw some conclusions about the adequacy of our choices.

1 Introduction

1.1 Context

The growing dependency on computer systems has created a need for fault tolerant systems to ensure uninterrupted business activities. A system is fault tolerant if it can mask the faults so that the running application detects no failure (i.e. all services continue to operate according to their specifications). A classical way to make a critical server reliable is to replicate it on different sites of the distributed system. If sites fail independently, an invocation to a fault tolerant server will succeed even if some replicas have crashed. Replication can be provided in two different ways: passive or active. With a passive replication mode, only the primary replica executes the request; a logging mechanism is used for recording object state and messages. When the primary fails, a new one is elected; the recovery mechanism allows it to get the correct state. With an active replication mode, all replicas concurrently execute the request; this mode reduces latency induced by the crash of a replica and, when no failure occurs, it contributes to enhance the performances of the replicated server.

A group communication system allows a collection of related objects to be considered externally as a single logical entity. This paradigm is used to simplify replication management and construct fault-tolerant applications. A group communication system [14] includes different services used by group members to coordinate their activities. One of those services is called Atomic Broadcast; it ensures that all requests received by the group entity are delivered to each member in the same order. Another fundamental service is the Group Membership. This service tracks changes in the group composition, due to the desire of members to join or leave the group, or the occurrence of crashes. The View Synchrony Communication allows the delivery of requests “in synchrony” with the delivery of changes in the group composition.

1.2 A Fault Tolerant CORBA Architecture Based on Agreement Components

**OPEN EDEN** is a fault tolerant CORBA architecture based on agreement components. A critical CORBA service is made reliable by replicating a CORBA object on different sites of the distributed system. The set of replicas forms a CORBA object group. **EDEN** provides a group communication system to **OPEN EDEN**, using agreement components from the ADAM [15] library. In the **EDEN** system, Atomic Broadcast, Group Membership and View Synchrony are considered as agreement problems, which can be reduced to a basic problem, called the Consensus problem. Through the use of a “generic agreement component”, implementing a generic consensus algorithm, and through some adequate components compositions, **EDEN** provides an elegant and modular way of implementing the fundamental services of a group communication system.

The design of **OPEN EDEN** has been driven by the desire
to use only portable techniques to implement the interaction between the EDEN system and CORBA. As a consequence, we decided to refuse solutions such as OCI [16] which uses proprietary interfaces to integrate a group communication system within an ORB. It provides an elegant design but is not portable across ORBs. We rather selected portable interceptors [1, 6], a mechanism specified by the OMG that allows to transparently add service code to the ORB. Compared to other approaches based on system-level interceptors (like in Eternal [5]), our approach is more portable since it can be easily migrated to different architectures. Transparency (for the application programmer) is another benefit of our approach; requests to a replicated service are transparently intercepted and redirected to the EDEN group communication system.

The objective of this paper is to describe our experience in using portable interceptors for interconnecting a CORBA platform with the EDEN group communication system and to draw some conclusions about the adequacy of this technique. A prototype of OPENEDEN has been implemented using the ORBacus [2] platform. The client part is written in Java and the server part is in C++.

1.3 Plan

The remainder of this paper is structured as follows. Section 2 provides an overview of two CORBA specifications: Portable Interceptors and Fault-Tolerance. Section 3 presents our system model. Section 4 describes the EDEN group communication system. Section 5 presents the architecture of OPENEDEN and gives some details about the implementation. Section 6 draws some lessons from this implementation. Section 7 discusses the advantages and drawbacks of our approach compared to other FT CORBA implementations. Finally, Section 8 concludes this paper.

2 Fault Tolerant Computing with CORBA

CORBA (Common Object Request Broker Architecture) is an architecture for building distributed objects. It is well suited for creating distributed applications running over heterogeneous hardware and operating systems. CORBA provides location transparency and programming language independence to application programmers. Despite these attractive features, CORBA has suffered from its lack of support for fault-tolerance. A first response to this problem has been given by the standardization of the transaction service (OTS) which aims at ensuring the ACID properties of a transaction. An alternative solution to fault-tolerance in CORBA consists in replicating objects on multiple hosts; this is addressed by the FT CORBA specification [6]. Comparing these two strategies is outside the scope of this paper; for an interesting presentation of both strategies and an elegant way of reconciling them, please refer to [12].

This paper focuses on object replication. Adding such functionality to existing ORBs requires either (1) to modify the ORB code or (2) to extend it with appropriate techniques. We put aside the first solution as it is too intrusive and highly dependent on the ORB implementation. We decided to use CORBA portable interceptors to implement the second solution. Portable interceptors provide a convenient way by which extension code can enhance an existing ORB without modifying it.

2.1 Portable Interceptors

Portable interceptors are hooks into the ORB through which ORB services can intercept the normal flow of execution of the ORB. The Portable Interceptor (PI) specification [6] defines IOR (Interoperable Object Reference) interceptors and request interceptors. The former allows to add specific components inside an object reference during its creation. The latter allows to intercept the flow of a client invocation to a server at different points; these points are indicated and numbered in Fig. 1. Some of these points are located at the client side and others are located at the server side.

Figure 1. CORBA Portable Interceptors

A client request PI allows to execute service code before the request is sent to the server (point 1 on the figure) and before the reply is returned to the client code (point 5). Using a specific CORBA exception mechanism, the interceptor code can change the target of an invocation. A server request PI allows to execute service before delivering the request to the server object (points 2 and 3) and after the server object has completed its job (point 4).

2.2 FT CORBA

In response to a Request For Proposal (RFP), issued by the OMG, regarding support for Fault Tolerance in CORBA, an OMG Final Specification has been adopted by the
OMG board; this specification has been integrated in the 3.0 release of the CORBA/IIOP document [6]. Figure 2 shows a simplified view of the architecture of a CORBA Fault-Tolerant system as proposed in the specification. The FT CORBA specification defines a set of standard interfaces for replication management, fault management and logging/recovery management. One of the main components of the FT CORBA architecture is the replication manager; its interfaces allow to specify the fault tolerance properties of an object group and provide operations for creating and deleting object groups (see Fig.2). The replication style is one of the FT properties for which three values are possible: COLD_PASSIVE, WARM_PASSIVE (passive replication is based on the use of a primary replica) and ACTIVE (all replicas play the same role in the group).

![Figure 2. FT CORBA](image)

One important point in the specification is the structure of an interoperable object group reference (IOGR). Such reference must be built according to a standard schema in order to allow inter-operability between client and server hosted by heterogeneous infrastructures. The specification defines the use of multiple TAG_INTERNET_IOP profiles to identify replicas (or a set of gateways) and introduces a new component TAG_FT_GROUP to encapsulate an object group identifier. An object group is identified by a Group IDentifier (GID) which is not related to the group membership. At group creation, the replication manager allocates the GID for the group and then constructs an object group reference populated with this GID.

The specification does not define a standard multicast group communication protocol to be used for maintaining consistency between replicas. Consequently, there is no inter-operability between ORBs hosting server objects; all the replicas of a server object group must be hosted by the same FT infrastructure. This particularity allows FT CORBA designers to choose the most appropriate protocol for their infrastructure.

## 3 System Model

As explained in the introduction, the OPEN EDEN system relies on a fault-tolerant consensus algorithm implemented in the EDEN group communication system. EDEN assumes an asynchronous system where nodes fail only by crashing. It also assumes that underlying failure detectors are available. These failure detectors are in charge of suspecting faulty replicas in the group. Unfortunately, due to the impossibility to safely distinguish crashed processes from a slow process or from a process with which communications are slow, it is impossible to implement perfect failure detectors. Our solution assumes an underlying failure detector of the class $\Diamond S$. $\Diamond S$ is the weakest class of failure detectors that allows solving the consensus in an asynchronous system. This class is defined by the following completeness and accuracy properties: Any process that crashes is eventually suspected (strong completeness) and there is a time after which there is a correct process that is no longer suspected (eventual weak accuracy). Relying on $\Diamond S$ failure detectors to solve agreement problems assumes that a majority of processes within the group never fails.

## 4 The Eden Group Communication System

EDEN is a group communication system which aims at developing reliable distributed applications. As illustrated by Figure 3, EDEN is built by configuring, assembling and deploying agreement components from the ADAM [15] library. These components rely on EVA [13], an event-based framework for structuring distributed services; EVA implements a publish-subscriber communication environment to structure entities composing high level protocols. In this architecture, protocols are regarded as a number of cooperating objects (entities) that communicate via an event channel.

Most of the reliable distributed abstractions of interest in fault-tolerant computing (Atomic Broadcast, Membership Management, Leader Election, Non-blocking Atomic Commitment, ...) are agreement problems. With regard to all these problems, processes belonging to a same group have, from time to time, to reach an unanimous decision. For example, in the Atomic Broadcast problem, the messages sent to the group have to be delivered in the same order to all the members. Due to the fact that messages are not received in the same order by all the processes, determining the delivery order can be viewed as an agreement problem. To construct this common global knowledge, all (non-crashed) members of the group have to repeatedly reach new agreements
Figure 3. Eden

to unanimously order the new messages. Several different agreement problems can be reduced into a basic problem called the Consensus problem. In particular, a solution to the consensus problem can be used as a building block to solve the reliable distributed abstractions mentioned above.

The central element of the ADAM library is the GENERIC AGREEMENT COMPONENT (GAC) [15]. It implements a generic and adaptive fault-tolerant consensus algorithm that can be customized to cope with the characteristics of the environment. Moreover, thanks to a set of versatile methods, its behavior can be tuned to fit the exact needs of a specific agreement problem. A range of fundamental ADAM components are implemented as specializations of this (GAC) component. By composing some of the ADAM agreement components, we can build a group communication logic. An ACTIVE REPLICATION service has been designed and forms the heart of the EDEN system. It is mainly based on the ATOMIC BROADCAST, GROUP MEMBERSHIP and GENERIC AGREEMENT components.

The ACTIVE REPLICATION service provides to OPEN EDEN all facilities needed to manage in a consistent way the CORBA object group. The components organization is not exposed to CORBA, which allows to maintain an independence between CORBA and the group communication system. The interactions between these two worlds are detailed in the next section.

5 The Open Eden Architecture

Our goal in designing the architecture of Open EDEN was (1) to obtain a portable and flexible implementation of the Fault Tolerant CORBA specification and (2) to experiment the usability of EDEN as a group communication support for such a service. For the sake of simplicity and because we did not focus on a full implementation of the FT CORBA specification, we decided the following restrictions:

- Passive replication is not supported in OPEN EDEN. Logging and recovery management mechanisms are not implemented.
- Membership and consistency are controlled by the infrastructure. It means that the fault-tolerant infrastructure is responsible for creating/deleting replicas and for ensuring the consistency of replicas. Application-controlled membership and consistency are not supported.
- Fault detection is not accessible via CORBA interfaces.

The rest of this section is organized as follows. Subsection 5.1 motivates our design. Subsection 5.2 presents the whole architecture of OPEN EDEN; it describes how a CORBA client can transparently invoke a group and how consistency is maintained at the server side. The management of group membership is also explained.

5.1 Design Choices

The FT CORBA specification defines the main components of a FT CORBA architecture, specifies their interfaces and precises the properties associated to an object group. However, the designer of a FT CORBA architecture is free in choosing the way the components are implemented and the way they interact. Regarding OPEN EDEN, we had to answer the following question “how do we connect two different worlds, CORBA and EDEN, which have been designed independently?”. As exposed previously, our requirements argue for a solution that is non intrusive from the CORBA point of view but also from the EDEN point of view. Therefore, we propose a solution based on the combination of an interception technique and a service approach.

Interception As explained in Section 2.1, a server interceptor allows to execute services before delivering a request to an object and also before returning a reply to a client. We use this capability to interconnect the CORBA world with the EDEN world. The idea is simple: before delivering
a request to an object, the interceptor asks to the **ACTIVE REPLICA TION** component of **EDEN** to order the request. The **EDEN** component decides the rank for this request (using a consensus protocol) and returns the information to the interceptor which can serialize the execution of all the requests.

**Service** The interface of **EDEN** has been written using the IDL language. In that sense, **EDEN** is considered as being a CORBA service. It has not been necessary to modify **EDEN** to connect it to the CORBA world. The **EDEN** service is used by the server interceptor to re-order the requests and also by the CORBA replication manager for membership management.

**OPEN EDEN = Interception + Service** Our solution combines the use of portable interceptors with a service approach. Interception is used to redirect the requests towards the CORBA **EDEN** service. This choice favors transparency with respect to the client code, portability with respect to an ORB implementation and is also CORBA-compliant.

### 5.2 Architecture

Figure 4 shows the architecture of Open **EDEN**; both client-side and server-side are presented.

**5.2.1 Client Side**

At the client side, we use (1) a gateway which acts as an intermediary object on the request path between a client and a server object group; (2) a client request portable interceptor component (CRPI), attached to a client and used to redirect client requests to a gateway.

The client of a server object group obtains the object group reference through usual ways, i.e., either by using a call to the CORBA Naming Service or by reading it from a file. Then, the client transparently (i.e., without knowing that the target object is an object group) invokes the method of the server object group. At this point, a client interceptor intercepts the call and extracts the following information from a tagged component: the identification of the gateways and the group identifier. Finally it redirects the request towards one of the available gateways.

As represented in Figure 4, the protocol used by the gateway to address an object group can be either IIOP or M-CAST/IIOP. In the former case, the group invocation is accomplished by using multiple unicasts. In the latter, the invocation is unreliably multicast to the group. We will not give details in this paper on how the gateway obtains the multicast reference (or list of references); in our prototype, the mapping between the object group identifier and references is implemented by the replication manager.

**5.2.2 Server Side**

A Server Request Portable Interceptor (SRPI) is attached to each server hosting a replica; it is used to implement the interface with the **EDEN** system.

Infrastructure-controlled consistency is one of the main requirements of the FT CORBA specification. As stated in the specification, it means that for the active replication style, "at the end of each method invocation on the object group, even in the presence of faults, the members should have the same state". To maintain group consistency, it is necessary that (1) each member behaves deterministically and that (2) the same sequence of requests be delivered in the same order to each member of the group. Regarding the first condition, like Narasimhan in [4], we consider that the unique source of undeterminism is due to potential concurrent executions of threads; we provide a specific scheduler to solve it. For the second condition, thanks to the **EDEN** system, we built an adequate delivery protocol. In the following paragraphs, we discuss strategies we used to (1) re-order the requests, (2) to ensure their reliable delivery and (3) to finalize their execution. All these strategies are implemented in a Server Request Portable Interceptor (SRPI) which interfaces CORBA with the **EDEN** system. The **SR-**
Pl component will use all the facilities provided by EDEN to reliably deliver the requests and to stabilize their reception.

To minimize the amount of information exchanged between the two worlds (CORBA and EDEN) and to optimize the performance of the protocols inside the EDEN system, the SRPI components will feed EDEN only with requests identifiers (instead of whole requests). The consequences of this choice are also examined in the following paragraphs.

Re-ordering Requests Each time a client request is intercepted, its thread is blocked and the SRPI component proposes its identifier (request identifier) to the EDEN system. At each site, EDEN collects the proposed identifiers and runs a consensus protocol with its peers to agree on a set of identifiers and a single delivery order for them. When a decision (or a set of decisions) has been taken, the EDEN system calls the SRPI components and gives them an ordered list of identifiers; each SRPI component can then mark the corresponding requests as deliverable. To protect against concurrent threads executions, a scheduling algorithm is implemented by each SRPI component to serialize the execution of the deliverable requests by their associated threads.

Reliable Delivery Channels used by clients to multicast requests are unreliable, so the reception of the requests by all the SRPI components are not ensured. Besides, as precised in section 3 (System Model), to decide for a delivery order, at least a majority of correct replicas must participate to the consensus. This necessary and sufficient condition to reach consensus leads us to two problems regarding the reliable delivery of requests. (i) Due to the necessity condition, EDEN can be blocked, waiting for a majority of proposals from the SRPI replicas. (ii) Due to the sufficient condition, EDEN can decide for the delivery of requests not yet received by the associated SRPI component. To deal with these problems, we added some specific operations to the CORBA EDEN interface. To solve problem (i), EDEN will ask the SPRI component to forward the set of requests it has proposed to the components that have not yet received them. To solve problem (ii), EDEN will call the associate SRPI passing it the taken decision and the list of SRPI components able to forward it the missing requests. Therefore, to assure the retransmission of messages, the SRPI components will put each new incoming request in a log before proposing its identifier to the EDEN system.

Stabilizing Messages When all SRPI components have proposed a same set of request identifiers, it is no more necessary for the SPRI components to keep a log of the corresponding requests. This situation is detected by the EDEN system which calls a specific operation of each SRPI component to ask them to remove requests from the log.

5.2.3 Group Membership

The FT CORBA specification states that the replication manager is in charge of adding/removing replicas to/from a group. In the EDEN system, the MEMBERSHIP MANAGEMENT components manage the membership of a group; they use a consensus algorithm to offer a consistent view to all members. We designed a CORBA interface which allows the CORBA replication manager to interact with a MEMBERSHIP MANAGEMENT component; this interface offers a join operation (called when a new replica has to join the group), and a leave operation (called when a new replica has to leave explicitly). Conversely, the replication manager is also noticed by the EDEN system when there is a change in the membership of a group, due to failures; an interface allows the MEMBERSHIP MANAGEMENT component to ask the replication manager for the installation of a new view.

6 Lessons Learned

In this section, we discuss about the benefits and drawbacks of using portable interceptors to connect CORBA with the EDEN group communication system.

Redirecting Client Invocation Due to known limitations of client portable interceptors [1], such as the impossibility to generate a reply or to filter among multiple replies, it is not possible to build a FT CORBA client without the help of an intermediary object. That is the reason why we use a gateway to redirect client invocation to an object group. The gateway, which uses the CORBA dynamic interfaces, DSI and DII, behave like a server for the client and like a client for the server object group.

Generating Object Group Reference Creating a group reference is an operation driven by the replication manager. When it is being asked to create a server object group by an application, the replication manager needs (1) to create the replicas by calling the factory operation create_object and (2) to generate a reference identifying the group. Unfortunately, since the replication manager is a regular CORBA object without any access to the internal ORB interfaces and functionalities, it has very few means for manipulating references. The only way to act, at a service level, on the creation of a reference is to use IORInterceptors. IORInterceptors only enable a server to add components. Thus, using IORInterceptors is insufficient for a replication manager because: (1) IORInterceptors do not provide for adding profiles, only components; (2) IORInterceptors are not invoked for already existing references, only for newly created ones; and (3) even for newly created references, it is underspecified when IORInterceptors are invoked.
Language Mapping We implemented the server part of our architecture in C++. In the C++ mapping, the interceptor is allowed to access all the attributes of the request structure. This way, we could get the arguments and log a request such that the SRPI component be able to forward it to others when necessary. Unfortunately, with the Java portable bindings, the arguments are not available. Consequently, we could not port the server part of our solution using the Java language.

Thread Model Dependency The strategy we used to reorder CORBA requests is strongly related to the concurrency model selected at the server-side and to the way portable interceptors are implemented. Using a “thread-per-request” model at the server-side and considering that, in ORBacus, interceptors are implemented in a manner such that the thread executing the interceptor code also executes the user server code, we managed to implement a scheduling policy that re-order requests (based on the Eden decisions). Any modification to the implementation of portable interceptors in the ORB (still compliant to the specification) could lead us to reconsider our implementation. Consequently, even if we have used only portable mechanisms (like portable interceptors) to implement requests re-ordering, we need to carefully examine how interceptors are implemented before migrating to a new ORB.

7 Related Work

Over the recent years, there has been much work on the different ways of implementing fault tolerance in CORBA. Some of these efforts adhere very closely to the FT CORBA specification ([5, 18]); others are voluntarily less FT CORBA compliant and favor innovative solutions ([3]).

Eternal [5] is based on the interception approach. Interceptors are implemented at the system level; using the /proc Unix interface, IIOP messages are intercepted and redirected to the Totem [19] group communication system. The advantage here is that Eternal can be used with any CORBA compliant ORB. The main drawback is the dependency on the Unix operating system.

IRL [18] is a three-tier replication logic; it has been used to prototype a FT CORBA compliant platform. The idea behind IRL is to distinguish three tiers. The first tier is the client; the second (middle) tier implements the replication logic: the last tier corresponds to the server replica. Portable interceptors are used to redirect client requests to the replication logic. The consensus protocol is run only at the middle tier; the sites where are located the replicas are not involved in the consensus protocol. This clean separation improves the scalability of the IRL system.

OGS [17] is based on the service approach. Clients use untyped interfaces and explicit invocations to the OGS service. With this scenario, clients are aware of the replication of the service and transparency is not provided. OGS also provides typed interfaces and transparent invocations using the dynamic capabilities of CORBA (DII and DSI); in that case, OGS compares to our gateway-based approach. However, the coupling between the group service and the CORBA platform seems to be stronger than in our system.

DOORS [20] is another partial implementation of the FT CORBA standard. Like OGS [17], DOORS is based on the service approach. However, DOORS does not implement active replication and does not provide a group communication system. Fault tolerance is provided through passive replication of objects.

FTS [3] proposes the use of portable interceptors and a specific group object adaptor (GOA) for handling replicated objects. Replication management and associated services (replica consistency, group membership) are implemented at the GOA level. This approach is interesting as it delegates to the GOA all functionalities which are difficult or impossible to implement at the portable interceptor level. The drawback here is that FTS requires the development of a specific GOA which is not portable across CORBA implementations. FT CORBA compliance is also partial.

8 Conclusion

This paper has presented OPEN EDEN, an implementation of the Fault Tolerant CORBA specification based on the use of a group system, called EDEN. Our goal was to prototype a FT CORBA service by using the most portable features of CORBA to interact with EDEN. We can summarize the results of this exercise by saying that client portable interceptors are useful tools to redirect client invocation and that server portable interceptors are also useful but not sufficient to provide all required functionality at the server side.

We are now investigating the design of a new architecture in which the EDEN system is still cleanly interfaced with an ORB but in which interfaces could be exposed to application programmer. Of course, in that way, we loose portability but one of our goals is now to bring to developers of fault-tolerant CORBA applications the benefits of the modularity of the EDEN system. For instance, in our prototype, we considered only infrastructure controlled consistency. Application controlled consistency is another possibility allowed by the FT CORBA specification. In that case, it could be interesting to provide to the application developer high-level services like consensus to help it to customize specific consistency protocols.
References


