Proceedings of the 2nd *International Symposium on Distributed Objects & Applications (DOA 2000)*, Antwerp, Belgium, 2000.

An Interoperable Replication Logic for CORBA Systems

Carlo Marchetti Massimo Mecella Antonino Virgillito Roberto Baldoni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy

{marchet,mecella,virgi,baldoni}@dis.uniroma1.it

Abstract

The Replication Logic is a set of protocols, mechanisms and services that allow a CORBA system to handle object replication. In this paper we present a specific implementation of a Replication Logic, namely Interoperable Replication Logic (IRL), which exhibits nice properties like non-intrusion (i.e., the replication logic is built "above" the ORB) and interoperability (i.e., a client can reside on an ORB while server replicas reside on other, possibly distinct, ORBs). We compare IRL to other CORBA systems implementing replication logic such as Eternal, OGS, DOORS, Isis+Orbix, etc.

Keywords: Object Replication, CORBA, Interoperability, Fault-Tolerance, Middleware, Distributed Systems

1. Introduction

A distributed application is fault-tolerant if it can be properly executed despite the occurrence of faults. Many different classes of distributed applications may require fault-tolerance, such as air traffic control systems, e-commerce applications, WEB services, telecommunication systems, etc. However, such applications can need different levels of reliability, availability and replicated data consistency. For example, stateless services like replicated web servers, need, at client side, simple failover mechanisms, stateful services, like telecommunication ones, require no downtime of the service along with strong data consistency.

Fault-tolerance can be obtained by software redundancy: a service can survive to a fault if it is provided by a set of server replicas. If some replicas fail, the others can continue to offer the service. At this end, servers can be replicated according to one of the following replication techniques: active replication or passive replication (primary-backup approach) [9]. The *Replication Logic* is the set of protocols, mechanisms and services that allow a distributed system to handle object replication.

The Common Object Request Broker Architecture (CORBA) [15][18] is a standard for object oriented distributed applications. It consists of a middleware on top of which applications can be designed, implemented and deployed in a very easy way. However, CORBA did not provide any tool for enhancing the reliability of such distributed applications. This had two major consequences:

- Many CORBA systems added replication logic to standard ORBs to cope with object failures and site crashes, like Eternal [13], OGS [6][7], DOORS [3], Isis+Orbix [12], Electra [12], AQuA [4] just to name a few.
- The Object Management Group (OMG) issued a RFP in 1998 that produced, in early 2000, the Fault Tolerant CORBA specification [5]. FT CORBA actually embeds many ideas coming out from the experience of previous systems and, from an operational viewpoint, it provides a set of IDL interfaces to an infrastructure implementing the replication logic.

In this paper fault-tolerant CORBA systems will be compared according to (i) the "intrusiveness" of the replication logic with respect to the ORB, (ii) the application object interoperability, i.e., the capacity of carrying out a client-server interaction when the client resides on an ORB while server replicas reside on other, possibly distinct, ORBs. Then we present an "Interoperable Replication Logic" (IRL), currently under development in our department, which has been designed following a non-intrusive approach with respect to the ORB and, to get interoperability, it has been designed "above" the ORB. Moreover, IRL has been designed to be FT CORBA compliant.

This paper is organized as follows. Section 2 introduces the notion of replication logic, of (non-) intrusive design and of interoperability. It also includes a short presentation of FT CORBA and its relationship with the replication logic. Section 3 presents the "Interoperable

Replication Logic": the architecture, the communication protocols and the main mechanisms for handling fault-tolerance. The same section contains a discussion about the main differences between IRL and other fault-tolerant CORBA systems. Section 4 concludes the paper.

2. Replication Logic

In this section we present the notion of replication logic, then we compare CORBA systems that have been proposed in the literature with the aim of adding availability and reliability to CORBA. These systems are compared with respect to: (i) how the ORB and the replication logic interacts and (ii) how they support interoperability among application objects in an heterogeneous multi ORB environment. Finally, we present the recent FT CORBA specification and how it relates to the notion of replication logic.

2.1. Replication Logic definition

Fault-tolerance can be obtained by software redundancy using one of the following replication techniques: active replication or passive replication (primary-backup approach).

In active replication a client sends a service request to a set of deterministic replicas, and waits for a certain number of identical replies. This number depends on the type of faults a replica can exhibit and on the consistency criterion we want to ensure on replicated data, i.e., the desired level of availability, consistency and reliability of the service. As an example, a stateless service (no data consistency is required) needs only a failover mechanism. Hence, the client can return the result when receiving the first reply. If replicated data consistency is required, at least the majority of replicas has to be aware of the request done by the client. If replicas can exhibit an arbitrary behaviour [1], the number of replicas that should be aware of the request goes to two-thirds.

In passive replication, a client sends a request only to a particular replica, namely the primary. Replicas, different from the primary, are called backup replicas (backups). If the primary fails, the backups elect a new primary. When the primary receives a request, (i) it performs the service, (ii) multicasts a message to the backups to notify the occurrence of the service and the update of replicated data and (iii) it sends back the reply to the client. If the service is stateless, step (ii) is not required.

We want to remark that both active and passive replication require several mechanisms in order to work properly: a reliable multicast protocol, a failure detection mechanism and an agreement protocol. The multicast and the agreement protocols can be combined to provide a total order multicast primitive that can be used in active replication to have a consistent evolution of the deterministic replicas. The failure detection mechanism and the agreement protocol can be used, in passive replication, to detect when a primary fails and for the election of a new one. Moreover, in both replication schemes, the usage of all these mechanisms can provide the nice notion of group abstraction with the relative services such as group abstraction, membership, state transfer, etc. From an operational point of view, these mechanisms and protocols can be implemented by group toolkits [17], such as Isis [2], Ensemble/Maestro [10][19], Totem [14], etc.

We call replication logic the set of protocols, mechanisms and services that have to be used in order to implement a replication technique, i.e., to offer fault-tolerance of a specific entity.

2.2. Intrusiveness

Fault-tolerant CORBA systems can be classified according to the degree of "intrusiveness" of the replication logic with respect to the standard ORB (Figure 1)[1]:

- **Intrusive design**: *a system is intrusive if its design requires to embed a part (or all) of the replication logic inside the ORB*. Intrusive systems differ on how many components of the replication logic are embedded into the ORB ("*deep*" vs. "*shallow*" intrusion).

- **Non-intrusive design**: *a system is non-intrusive if its design decouples the replication logic from the ORB.* Non-intrusive systems differ on whether the replication logic is located *"above"* the ORB (i.e., the replication logic exploits only ORB features) or not. In the latter case we say that the replication logic is *"below"* the ORB[2].

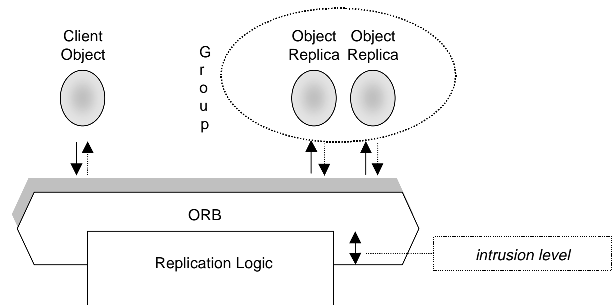

Figure 1. Intrusive and non-intrusive design

---

1 A previous classification of fault-tolerant CORBA systems was proposed by Felber et al. [8]. This taxonomy classifies approaches into *integration*, *interception* and *service*, according to some architectural issues and service specification aspects.

2 Note that a non-intrusive replication logic that uses even a single mechanism which is not provided by the ORB is considered "below" the ORB.

Intrusive design has been followed by systems as Orbix+Isis and Electra [12] ("deep" intrusion), and AQuA [4] ("shallow" intrusion).

Orbix+Isis has been the first commercial product offering fault-tolerance and high availability to CORBA compliant applications. The Orbix ORB core [11] has been modified in order to distinguish between invocations to object groups and to standard objects. These invocations are handled by the Isis group toolkit [1] and by the Orbix ORB core respectively. Moreover, the modified ORB exploits functionalities offered by the Isis group toolkit such as process group creation/deletion, reliable multicast, ordering of events, fault monitoring, all embedded in a virtual synchrony model. A CORBA object, in order to become member of a group, overrides some methods inherited from a proprietary base class, and exposes these methods in its interface. These methods are invoked directly from the modified ORB both to perform specific action (i.e., state transfer, fault monitoring) and to notify events (i.e., view changes).

Electra differs from Orbix+Isis mainly for the possibility of integrating different group toolkits by rewriting an adaptation layer. In both systems replication logic is embedded both into the ORB (group abstraction) and into the group toolkit (multicast protocol, failure detection and agreement protocol). This makes the intrusion "deep".

AQuA achieves availability and reliability by replacing the ORB IIOP implementation with a proprietary gateway. This gateway wraps IIOP calls made by the ORB to object groups into invocations to Maestro/Ensemble group toolkit [10][19]. In this case the replication logic is handled by the gateway and by the group toolkit. As it is necessary only to rewrite the IIOP modules of the ORB, AQuA is a "shallow" intrusion system.

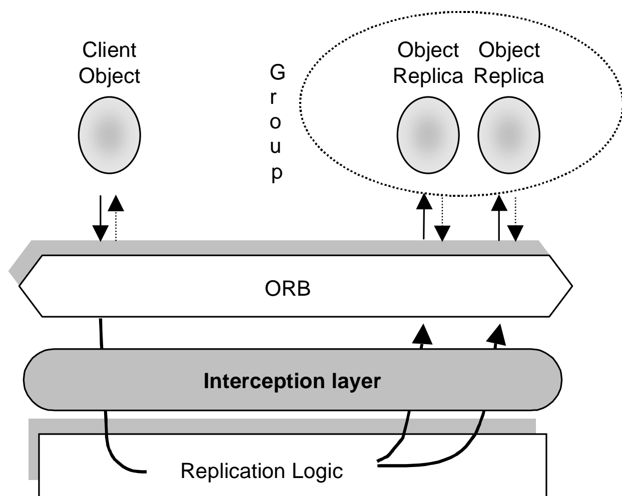

Figure 2. Replication logic "below" the ORB: the Eternal system

Non-intrusive design has been followed by systems as Eternal [13] (replication logic "below" the ORB), Object Group Service (OGS) [6][7] and DOORS [3] (replication logic "above" the ORB).

The rationale behind Eternal is to intercept all network system calls made to the OS and to distinguish between non-IIOP and IIOP messages. IIOP messages are forwarded to a group toolkit (e.g. Totem [14]). In Eternal the replication logic is entirely managed by the group toolkit while the interception of IIOP messages is done by a "thin" interception layer (Figure 2).

Contrarily to previous systems, OGS defines a Common Object Service (COS) [16] with IDL interfaces designed to invoke the functionalities of a group toolkit. It is a non-intrusive system whose replication logic is "above" the ORB (Figure 3). OGS is designed as a modular set of services, thus, a client can invoke the service it needs; OGS is not limited to handling fault tolerance: it can be exploited for load balancing, multiversion object management, fault monitoring and control.

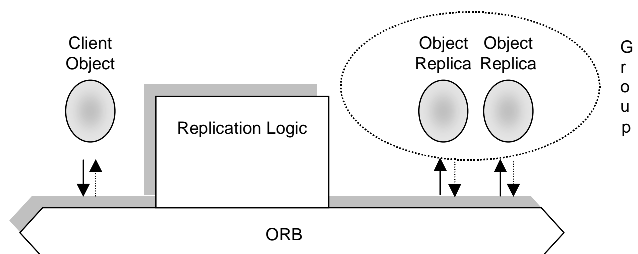

Figure 3. Replication logic "above" the ORB: OGS and DOORS systems

DOORS is a COS offering failure detection and recovery mechanisms to fault-tolerant applications. DOORS is implemented by two main components, namely the ReplicaManager and the WatchDog (a distributed failure detector). The former component handles group definition, replica consistency and uses the latter to detect failures and to determine when a recovery action on a set of replicas has to start.

2.3. Interoperability

With the term *interoperability* we mean *the possibility of run-time interactions of application objects deployed on top of distinct ORBs* (this notion does not include the issue of the interaction of different replication logics). This scenario is shown in Figure 4. In particular, we address the following issue: the replication logic is able to manage application objects running on top of distinct ORBs (possibly different from the one where the replication logic runs), or, conversely, the replication logic and

the application objects need to run on the same ORB platform. In the following, we give an idea of the cost that fault-tolerant CORBA systems have to pay to get such an interoperability. This cost takes into account the amount of modifications that application objects and the ORBs need.

Getting interoperability depends on how the interactions among remote objects (i.e., among server replicas, non-replicated objects and replication logic components) are implemented. A system is interoperable only if it uses as communication mechanisms the standard remote method invocations offered by the ORB via IIOP. As a consequence, non-intrusive fault-tolerant CORBA systems (e.g. OGS, DOORS) developed “above” the ORB are interoperable without any modification (i.e., free interoperability).

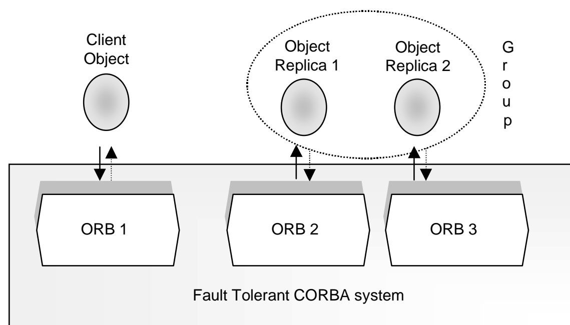

Figure 4. Interoperability in an heterogeneous multi ORB environment

Isis+Orbix and Electra (i.e., “deep” intrusive fault-tolerant CORBA systems) do not get interoperability as it is necessary to provide the same ORB (e.g. Isis+Orbix) to all computing resources.

AQuA (i.e., “shallow” intrusive fault-tolerant CORBA system) can be interoperable provided that we develop one gateway for each distinct pair <OS, ORB> included in the heterogeneous environment.

Eternal (i.e., non-intrusive “below” the ORB fault-tolerant CORBA system) requires to write a interception layer for each distinct OS.

This comparison is shown in the first row of Table 1 (see Section 3.7).

2.4. OMG FT CORBA specification

FT CORBA provides a standard and efficient way to develop fault-tolerant applications by object replication. It fixes two main properties that a generic FT CORBA compliant system has to ensure: strong replica consistency and consistent recovery.

In order to ensure these properties, FT CORBA *defines* an infrastructure, referred as Fault Tolerant Infrastructure (FTI), that acts as a middleware simplifying the development of dependable applications. From another perspective, we could say that FT CORBA defines some interfaces that a replication logic makes visible to CORBA objects. As a consequence, FT CORBA does not specify implementation aspects of the FTI (i.e., of the replication logic), as for example how to implement a multicast protocol, a failure detection mechanism, etc.

The specification defines:

- Some common features to be implemented by ORBs, in order to be able to interact with a generic FTI; these features are mainly concerned about transparent re-invocations of requests by client application objects and ORBs. Every client object can invoke simple (stateless) fault tolerant objects, without requiring any FTI to run on the server side.
- A set of architectural components and a collection of IDL interfaces, necessary to develop fault-tolerant applications on top of a generic FTI and to interact with the infrastructure. In particular, FT CORBA defines both some interfaces for the management of the replication logic implemented by the infrastructure (*infrastructure interfaces*), and other interfaces that have to be implemented by an object to be replicated and managed by the infrastructure (*object interfaces*). The infrastructure interfaces can be used by external administrative tools, e.g. to set the properties of a particular group (the replication technique, the minimum number of replicas, etc.). The object interfaces allow the infrastructure to monitor object failures and to read and write object state, in order to perform replica recovery.

3. Interoperable Replication Logic (IRL)

IRL is a software infrastructure on top of which reliable distributed applications can be built. IRL has been developed following a non-intrusive design “above” the ORB (Figure 5), thus remote objects interact through standard IIOP messages. IRL final target is to develop a FT CORBA compliant infrastructure.

In IRL, the replication logic has been centralized into a single logical component. Contrarily to other non-intrusive approaches residing “above” the ORB (i.e., OGS and DOORS), IRL does not define service interfaces (such as multicast communication primitives, failure detection mechanisms, etc). In IRL, a client interacts with a set of server replicas as they were “singleton” objects and server replicas are not aware of their replication (except they have to provide some interfaces). IRL offers interfaces for the management of the server replication (e.g. active or passive replication) and for the desired level of consi-

stency of replicated data. This approach is close to the FT-CORBA specification.

Currently, IRL supports active replication of server objects, running on different CORBA 2.3 compliant ORBs (interoperability), for two types of server application objects, namely stateful servers and stateless servers. The former are generic deterministic servers for which the reply to a request depends on the initial state and on the sequence of previously invoked requests. The latter are deterministic servers whose state is not modified by the method invocations.

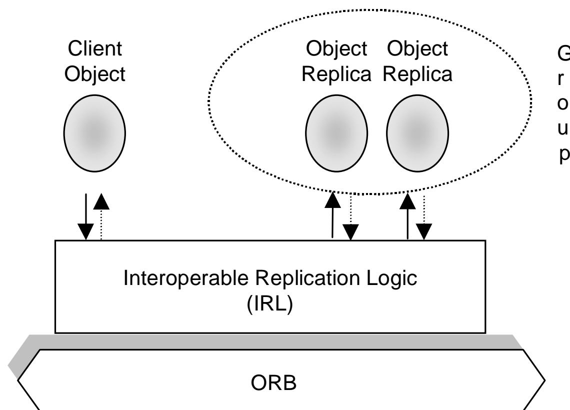

Figure 5. Replication logic "above" the ORB: the IRL design

3.1. Architectural Overview

IRL (Figure 6) is composed by a collection of different CORBA objects packaged into three main components, namely *IRL Core*, *SmartProxy* and *ServerRunTimeSupport* (ServerRTSupport, SRTS). IRL Core is composed by a set of CORBA objects running into a single process over a generic ORB. IRL Core implements the replication logic for both the application objects and IRL Core itself. In fact, in order to prevent IRL Core to be a single point of failure, it is replicated onto different hosts, using the passive replication technique. SmartProxy is a component residing above the client ORB that hides server replication to the client. ServerRTSupport contains a set of CORBA objects that has to be deployed on each host containing, at least, a running replica of a server application object. These objects allow to manage a portion of the failure detection and recovery mechanisms and to hide the active replication protocol details (such as request filtering) to the server object replica.

3.2. Overview of the IRL Core

IRL Core consists of three types of CORBA objects: a set of `ObjectGroup` objects, an `ObjectGroupManager` object and a `PersistenceManager` object.

An `ObjectGroup` object is logically associated to each set of replicas (i.e., a group) of a given application server object: it receives all the requests directed to a particular group, executes them on each replica and is responsible, in the case of a stateful server, for maintaining consistency among the replicated data managed by replicas. Moreover, an `ObjectGroup` object implements a portion of the failure detection mechanism (the other portion is implemented by the ServerRTSupport), it can trigger replica recovering and it can add/remove replicas to the group. To perform these tasks, each `ObjectGroup` object maintains the following information, denoted OGI (Object Group Information):

- the Interoperable Object References (IORs) of the object replicas and the information about their internal state and availability;
- the properties of a group, i.e., minimum number of replicas, the replication style, etc.

The addition/removal of replicas to/from a group can be done by external IRL management applications. As a consequence, the `ObjectGroupManager` object offers in its interfaces methods to execute such operations.

The `PersistenceManager` object is the component responsible for the replication of the IRL Core. IRL Core follows a passive replication paradigm, thus at any time it exists at most one primary IRL Core and possibly one or more backups. The `PersistenceManager` object on the primary IRL Core behaves as client of the `PersistenceManager` objects on the backups, notifying them each change in the internal state of the primary IRL Core (e.g. group membership modifications) through CORBA one-way invocations. Moreover, backup `PersistenceManager` objects monitor the primary IRL Core for failures. If a primary crashes, they start an agreement protocol to elect a new primary.

3.3. An overview of the client/server interaction

A client interaction with singleton objects follows standard CORBA specification. An interaction between a client and a group of replicas is based on three main mechanisms:

- indirect object group addressing;
- serialization of object group requests;
- multicast protocol formed by point-to-point communications.

Indirect object group addressing. A request to an object group is made by the client through a special type of IIOP 1.1 compliant IOR, namely OGR (Object Group Reference). OGR actually is a local reference to the SmartProxy but contains, in its TaggedComponent field, a reference to the `ObjectGroup` object to be invoked.

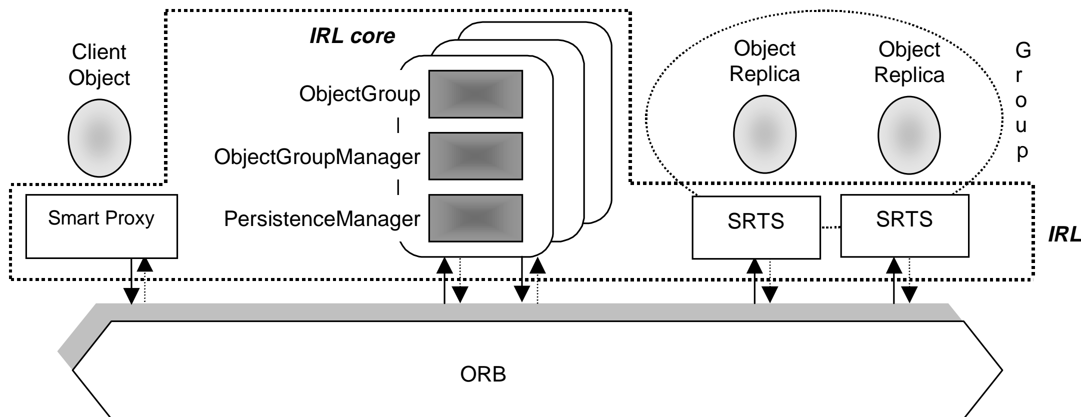

Figure 6. The IRL architecture

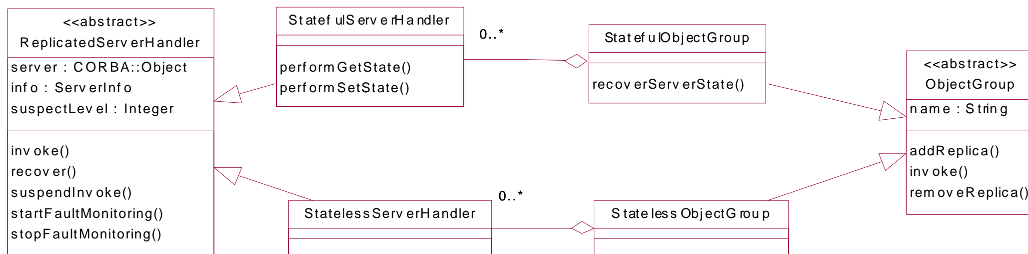

Figure 7. Class diagram of the `ObjectGroup`

Thus, requests are not directly invoked by the client on the `ObjectGroup` objects: the SmartProxy actually receives the request and forwards it to the primary IRL Core (indirect object group addressing). This 3-tier mechanism allows to hide the replication of the IRL Core to clients.

Serialization of object group requests. An `ObjectGroup` object acts as a serializer of requests incoming from distinct clients. It receives requests sent to its group and multicasts them to the group replicas. This simple engineering mechanism allows replicas to perceive the same sequence of requests incoming from distinct clients.

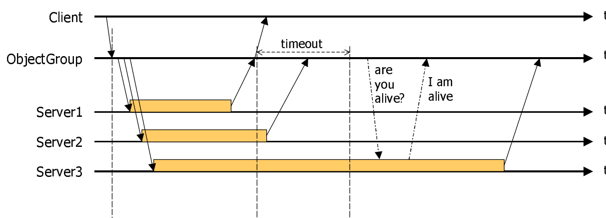

Figure 8. Client/server interaction

Multicast protocol. As we develop the protocol "above" the ORB, we assume reliable communication channels between `ObjectGroup` objects and the group replicas (this is actually the case as all interactions among objects are implemented as CORBA remote method invocations based on IIOP that is layered over TCP/IP). Moreover, we assume it always exists a correct replica in each group, i.e., a server that follows its specification.Consider Figure 8: when a request from a client arrives at the corresponding `ObjectGroup` object, if another request is being processed, it is queued until previous requests have been processed. Otherwise, the `ObjectGroup` object sends the request to the group replicas by using a point-to-point reliable communication, one for each replica. Then, the `ObjectGroup` object waits for the first reply. When it arrives, the `ObjectGroup` object forwards the reply to the client and keeps waiting for the others for a short period of time (to be set at run-time as a function of the round trip time). All the servers that do not reply within this period, are considered «suspected».If a replica is suspected for a certain number of times consecutively, the `ObjectGroup` object starts a fault monitoring process in order to decide if the replica should

be considered as «crashed»[3]. The fault monitoring process consists in periodically invoking an `areYouAlive()` operation on the ServerRTSupport associated to the replica. If the replica is neither crashed nor blocked and the communication network is not congested, it will reply before a certain amount of time, otherwise it will be considered faulty. In this case the `ObjectGroup` object removes the replica from the group and can take appropriate decisions according to the group properties. For example, if the number of active replicas is below the threshold defined when the group was created, the `ObjectGroup` object starts another consistent replica.

3.4. Client/server interaction: protocol design

Figure 7 illustrates the class diagram in UML notation of the `ObjectGroup` implementation. It shows the programming language objects that implements the functions. An `ObjectGroup` object exposes in its IDL interface the `invoke()` method, called by the SmartProxy to invoke a client request on the group.

The `ObjectGroup` class represents a group of replicated application server objects. It offers methods to multicast a particular request to the replicas belonging to the group (i.e., the `invoke()` method) and to manage the group composition (i.e., the `addReplica()` and `removeReplica()` methods). The `ObjectGroup` class is specialized by either the `StatefulObjectGroup` class or the `StatelessObjectGroup` class. Each replica of a server application object belonging to a group is associated to an instance of the `ReplicatedServerHandler` class. It maintains information about the location and the status of the remote server, and offers methods to invoke operations on the server, detect its failures and recover its state. IRL supports stateful and stateless servers. Stateless servers can be replicated without any modification, while stateful servers must implement the `getState()` and `setState()` methods required for their recovery. To recover a particular replica the `StatefulObjectGroup` object (i.e., an object group of stateful replicas) exposes the `recoverServerState()` method. This method uses the `performSetState()` and `performGetState()` methods of the `StatefulServerHandler()` objects. These methods uses the `getState()` and `setState()` methods of the associated replicas in order to respectively update and read the replica state.

A sequence diagram of a client/server interaction is shown in Figure 9, where standard intra-host programming language invocations are represented with straight lines and CORBA inter-host remote method invocations are represented with dashed lines. In the figure, S1 and S2 are two object replicas, RS1 and RS2 are the associated handlers.

We would like to remark that even if client/server interactions related to a single group are serialized by its `ObjectGroup` object, interactions involving distinct object groups can be concurrently executed by their `ObjectGroup` objects.

3.5. IRL Core Replication Design

In this section we point out how the basic mechanism of the passive replication technique have been embedded in the object oriented design of IRL Core.

The state of IRL Core can be identified as the information necessary to a backup to correctly takeover the primary role when a primary fails. In particular, this information is composed by:

- The number of `ObjectGroup` objects and their identifiers.
- For each `ObjectGroup` object:
  - the OGI (see Section 3.2);
  - the information about the current client/server interaction (if any).

As soon as the state of the IRL changes, an incremental state update is carried out by the primary IRL via one-way CORBA invocations, one for each backup. In Figure 10 is presented the UML class diagram of the programming objects devoted to accomplish the passive replication protocol of IRL Core.

Objects able to modify the IRL Core state are `Updatable` objects. `ObjectGroup` objects and `ObjectGroupManager` objects are `Updatable` objects i.e., they implements the `Updatable` interface. As the IRL Core is passively replicated, we have primary `Updatable` objects and backup `Updatable` objects.

A primary `Updatable` object triggers a state change by calling the `stateTranfer()` method of the IRL Core `PersistenceManager` object. The incremental state update information is stored in an instance of the `StateUpdateInfo` IDL structure. This structure is transferred to each backup `PersistenceManager` object by a one-way CORBA invocation of the `updateState()` method. This update is done by carrying out a multicast protocol using point-to-point reliable communication (as the one described in Section 3.3).

When a backup `PersistenceManager` object executes the `updateState()` method, it updates the state of its `Updatables` according to the information contained in the `StateUpdateInfo` structure.

[3] The failure detection is performed by an `ObjectGroup` object only during the execution of a request.

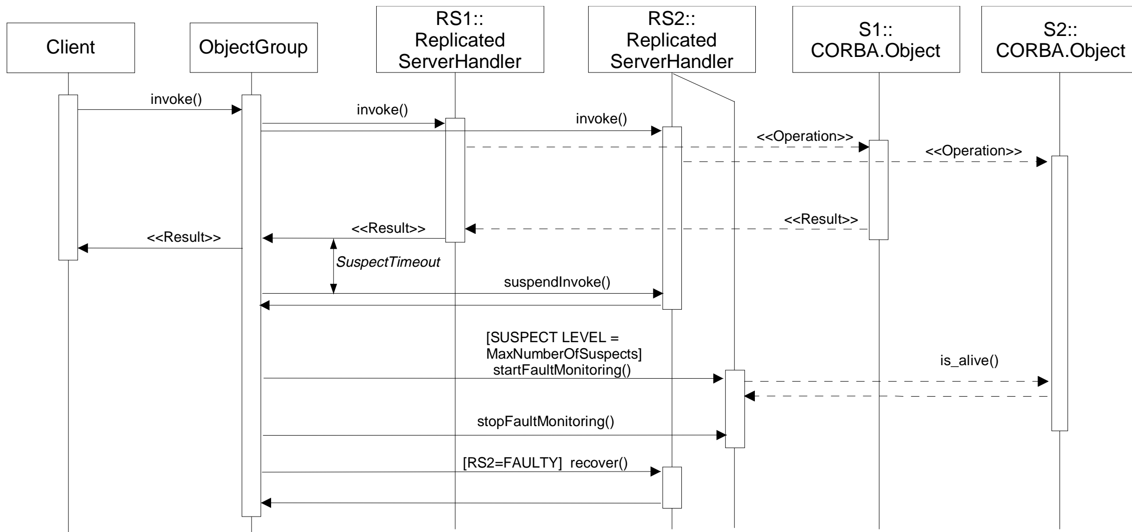

Figure 9. Sequence diagram of a client/server interaction

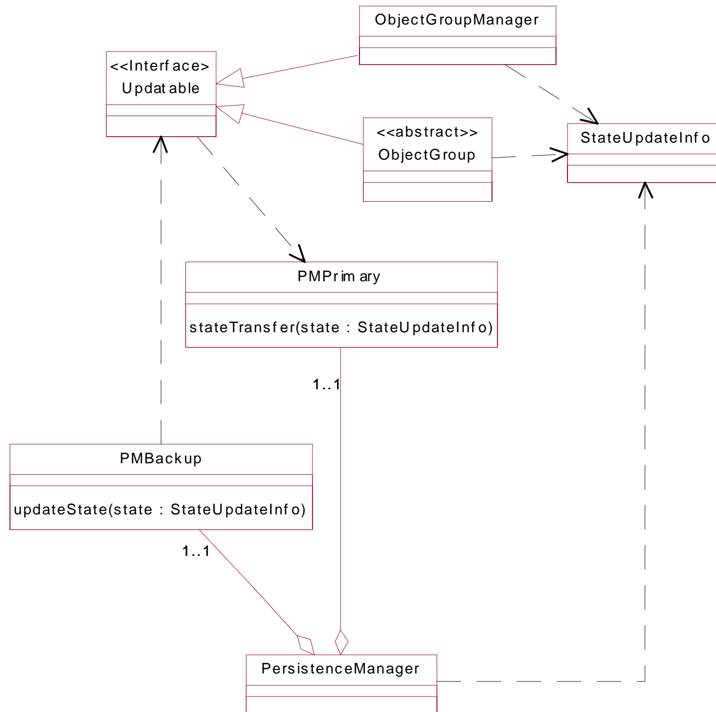

Figure 10. Class diagram of the `PersistenceManager`

	Intrusive		*Non-intrusive*		
	"deep"	***"shallow"***	***"below" the ORB***	***"above" the ORB***	
	(Isis+Orbix, Electra)	***(AQuA)***	***(Eternal)***	***(OGS)***	***(IRL)***
Interoperability (see Section 2.3)	Impossible	Very Expensive	Expensive	Free	Free
Replication Logic Implementation	Group Toolkit	Group Toolkit	Group Toolkit	Distributed	Centralized with Passive Replication
Replication Logic Visibility	Black box	Black box	Black box	White box (fine grained services)	Black box

Table 1. Comparison among fault-tolerant CORBA systems

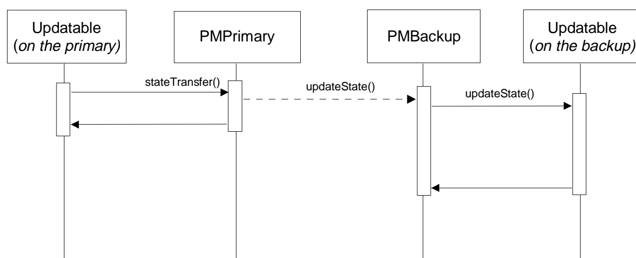

Figure 11. State change of the primary IRL Core

The sequence diagram shown in Figure 11 describes the interaction between the primary IRL Core and the backups.

3.6. A Remark on Consistency of Replicated Objects

When the IRL Core primary fails, a problem of consistency arises if a multicast protocol is incomplete (i.e. a request/update has been sent only to a subset of a group of stateful replicas). This situation can occur in two distinct cases: when the `PersistenceManager` object is running the multicast protocol described in the previous section and when an `ObjectGroup` object is running the multicast protocol described in Section 3.3.

In the former case, an agreement protocol is performed among the backups in order to elect an updated backup as new primary (i.e., a backup that received the last update). In the latter case, once a new IRL Core primary has been elected, each `ObjectGroup` object restarts an incomplete multicast (if any). If the replica already received the request, the local ServerRTSupport filters it out.

3.7. Discussion

Let us start this discussion by comparing IRL with other fault-tolerant CORBA systems according to three architectural issues, i.e., *interoperability* (see Section 2.3), *replication logic implementation* and *replication logic visibility*. This comparison is summarized in Table 1.

- *Interoperability*. IRL gets *interoperability,* such as OGS and DOORS, without any modification to its design, as it is based on a non-intrusive approach "above" the ORB.
- *Replication logic implementation*. Intrusive systems and Eternal rely on a specialized, proprietary group toolkit. Conversely, OGS offers a completely distributed replication logic implemented "above" the ORB. IRL centralizes the replication logic in its Core, providing dependability by passive replication.
- *Replication logic visibility*, i.e., what a distributed application "sees" of the services provided by the replication logic. A system is *"black box"* if its replication logic offers to applications only property management interfaces (i.e., the mechanisms implementing fault-tolerance are not accessible from outside); a system is *"white box"* if its replication logic is composed by a set of services offered to applications. IRL, Eternal and intrusive systems offer "black box" replication logics. OGS, instead, is a "white box" replication logic system: it offers interfaces to use a set of services. Note that the black box approach is the one adopted by FT CORBA specification (see Section 2.4).

Let us finally consider a more close comparison with OGS, which is historically the *first* non-intrusive system from which IRL has inherited some design ideas. However, IRL employs a centralized replication logic implementation (with passive replication), while OGS follows a fully distributed approach. This makes easier IRL implementation, but the centralization can be a performance bottleneck. As an example, requests sent to a group of replicas need to be serialized by the related `ObjectGroup` object; on the other hand, this serialization makes much easier to enforce the same order of request delivery to each replica. We plan to evaluate IRL

performance with respect to OGS in order to check if this performance gap really exists and to quantify its wideness.

4. Conclusions and Future Work

This paper has presented a novel replication logic (IRL) that allows a CORBA system to build dependable and reliable applications by using software replication. IRL is based on a non-intrusive design developed "above" the ORB. The paper has also shown a comparison with the more relevant fault-tolerant CORBA systems presented in the literature.

We are currently implementing the IRL design and adapting it to be FT CORBA compliant. As a future work we plan to add scalability to IRL by considering concurrent execution of IRL Cores. Each of them is passively replicated and manages a subset of the `ObjectGroup` objects running in the system.

Acknowledgements

The authors would like to thank Paolo Papa, who joined the IRL project during his master thesis.

References

[1] R. Baldoni, J.M. Helary, M. Raynal, "From Crash Fault Tolerance to Arbitrary Fault Tolerance: Towards a Modular Approach", to appear in Proceedings of *International Conference on Dependable Systems and Networks* (formerly FTCS), New York, NY, 2000.

[2] K. Birman, R. Van Renesse (eds.), *Reliable Distributed Computing with the ISIS toolkit*. IEEE CS Press, 1994.

[3] P. Chung, Y. Huang, S. Yajnik, D. Liang, and J. Shih, "DOORS - Providing fault tolerance to CORBA objects" poster session at *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, 1998.

[4] M. Cukier, J. Ren, C. Sabnis et al., "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects", in Proceedings of *the IEEE 17th Symposium on Reliable Distributed Systems (SRDS-17)*, West Lafayette, IN, 1998.

[5] Ericsson, Eternal Systems et al., *Fault Tolerant CORBA. Joint Revised Submission*. OMG TC Document orbos/99-12-19, Object Management Group, Framingham, MA, 1999.

[6] P. Felber, *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*. PhD thesis (no. 1867), École Polytechnique Fédérale de Lausanne, Switzerland, 1998.

[7] P. Felber, B. Garbinato, R. Guerraoui, "The design of a CORBA Group Communication Service", in Proceedings of *the 15th Symposium on Reliable Distributed Systems (SRDS-15)*, Niagara-on-the-Lake, Canada, 1996.

[8] P. Felber, R. Guerraoui, A. Shiper, "Replicating Objects using the CORBA Event Service", in Proceedings of *the 6th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems (FTDCS-6)*, Tunis, Tunisia, 1997.

[9] R. Guerraoui, A. Shiper, "Software-Based Replication for Fault Tolerance", in A.K. Somani, N.H. Vaidya (eds.), Special Section on Fault Tolerance, *IEEE Computer*, April 1997.

[10] M.G. Hayden, *The Ensemble System*. Ph.D thesis, Cornell University, Ithaca, NY, 1998.

[11] IONA Technologies Inc., *The Orbix system*. `http://www.iona-portal.com/suite/orbix.htm`.

[12] S. Landis, S. Maffeis, "Building Reliable Distributed Systems with CORBA", *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.

[13] L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, L.A. Tewksbury, V. Kalogeraki, "The Eternal System: an Architecture for Enterprise Applications", in Proceedings of *the 3rd International Enterprise Distributed Object Computing Conference (EDOC'99)*, Mannheim, Germany, 1999.

[14] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, C.A. Lingley-Papadopoulos, "Totem: A Fault-Tolerant Multicast Group Communication System", *Communications of the ACM*, vol.39, no.4, 1996.

[15] Object Management Group (OMG), *The Common Object Request Broker Architecture and Specifications. Revision 2.3*. OMG Document formal/98-12-01, OMG, Framingham, MA, 1998.

[16] Object Management Group (OMG), *CORBAServices: Common Object Services Specification*. OMG Document formal/97-07-04, OMG, Framingham, MA, 1997.

[17] D. Powell (ed.), Special Section on Group Communication, *Communications of the ACM*, vol. 39, no. 4, pp. 50-97, April 1996.

[18] R. Soley (ed.), *Object Management Architecture (OMA) Guide. Second Revision*. OMG Framingham, MA, 1992.

[19] A. Vaysburd, K. Birman, "Building Reliable Adaptive Distributed Objects with the Maestro Tools", in Proceedings of *Workshop on Dependable Distributed Object Systems, OOPSLA'97*, Atlanta, GA, 1997.