

# Design of an Interoperable FT-CORBA Compliant Infrastructure

C. Marchetti, A. Virgillito and R. Baldoni  
Dipartimento di Informatica e Sistemistica  
Università di Roma “La Sapienza”  
Via Salaria 113, 00198, Roma, Italy  
email: {marchet,virgi,baldoni}@dis.uniroma1.it

## Abstract

In this paper we present the design of IRL (Interoperable Replication Logic), a FT-CORBA compliant platform that provides transparent client-server interactions and server failovers to application clients by using a set of replicated CORBA objects (IRL components). As cooperation among IRL components is carried out using standard CORBA invocations, IRL allows deployments of CORBA server objects and IRL components over ORBs from distinct vendors satisfying the interoperability property.

## 1 Introduction

Many different classes of distributed applications require *fault tolerance*, that is the ability to continue their execution even after the occurrence of failures. Fault tolerance can be implemented in software following either the *active* or the *passive replication technique* [4].

Replication and failure management are usually handled by a middleware layer that provides transparent client-server interactions and transparent failovers to clients. However, one of the most popular object-oriented middleware standard, CORBA [12], until revision 2.4, did not provide any mechanism to handle object replication and failure management. Thus, many CORBA-compliant platforms (e.g. [1], [2], [5], [6], [9]) were developed to support object replication in CORBA applications. Taxonomies of these systems can be found in [3] and [6].

In 1998 the OMG issued a RFP ([10]) with the aim of standardizing a result of these efforts, i.e. of defining a single standard architecture embedding some of the ideas from the previous experiences. The result is the Fault Tolerant CORBA (FT-CORBA) specification [13], published in early 2000, that will be included in CORBA 3.0 specification. FT-CORBA defines a set of service interfaces and some enhancement to the standard ORB, in order to provide support for fault tolerance of CORBA-based distributed applications through redundancy of CORBA objects. A fault tolerance platform is FT-CORBA compliant if it exports the FT-CORBA specification interfaces and implements a subset of its functionality. At the best of our knowledge, two existing systems are FT-CORBA compliant: *Eternal* [8] and *DOORS* [9].

In this paper we introduce the Interoperable Replication Logic (IRL, [6], [7]) system design, whose implementation is currently carried out in our department. In IRL, CORBA invocations from clients to object groups are sent to IRL components, a set of CORBA objects providing active replication of CORBA server objects. IRL implements replication using standard CORBA invocations and is thus logically placed *above* the ORB. As a consequence, it exhibits the nice *interoperability* property: replicated application objects and IRL components can be deployed over ORBs from distinct vendors, allowing thus to overcome FT-CORBA vendor dependence limitation [13].

The IRL system takes also into account the following issues:

**FT-CORBA compliance.** IRL components exports interfaces defined in the FT-CORBA specification and have been implemented following the FT-CORBA mandatory requirement, i.e. “... *the absence of single point of failures* ... ” [13].

**IRL Component Replication Paradigm.** To fulfill previous FT-CORBA requirement, each IRL component is replicated according to a passive replication style.

The paper is organized as follows: in Section 2 the FT-CORBA specification is summarized and the IRL system design is presented: each IRL component is described with its functionality and its relationship with the FT-CORBA standard. Section 3 outlines how each IRL component has been rendered fault-tolerant. In Section 4 an example of a client-server interaction showing the architecture behaviour is given. Section 5 concludes the paper.

## 2 IRL FT-CORBA Compliant Design

In this section we first introduce a summary of the FT-CORBA specification, and then present the design of IRL (Interoperable Replication Logic), a FT-CORBA compliant interoperable fault tolerance middleware.

### 2.1 Overview of FT-CORBA Specification

In the FT-CORBA specification, fault tolerance is achieved through entity redundancy, fault detection and recovery. The replicated entity is the CORBA object, and replicated objects are managed through the abstraction of object group in order to guarantee strong consistency. To identify an object group, FT-CORBA introduces the *interoperable object group reference* (IOGR). An IOGR is an Interoperable Object Reference (IOR, [12]) composed by multiple *profiles*, each identifying an object group member (in case of passive replication) or a set of gateways providing access to the whole group (in case of active replication). IOGRs are used by FT-CORBA compliant client ORBs in order to provide clients with replication and failure transparency by the *transparent client reinvocation* and *redirection* mechanisms.

In order to support the object group abstraction and the life cycle of object groups, FT-CORBA extends CORBA with mechanisms, architectural components, IDL interfaces and with the so called *fault tolerance properties* (FTPs), embedded in an infrastructure called *Fault Tolerance Infrastructure* (FTI). Such mechanisms and components are organized into three features: *replication management*, *fault management* and *recovery management*.

*Replication management* consists in the creation/removal of object groups or object group members and in FTP modification. The **ReplicationManager** component is responsible for carrying out such activities, when requested through its interface (**ReplicationManager**).

*Fault Management* concerns the detection of object failures, the creation and the notification of fault reports and the fault report analysis. These activities are respectively carried out by the **FaultDetectors**, **FaultNotifier** and **FaultAnalyzer** components. Each replicated object can be monitored for failures by implementing the **PullMonitorable** interface. **FaultDetectors** invoke the **is\_alive()** method of such interface performing pull-style failure detection. If a replica does not timely return from this invocation, **FaultDetectors** assume a crash on the replica and push a fault report to the **FaultNotifier**. **FaultNotifier** receives fault reports from the **FaultDetectors** and propagates fault notifications to the **ReplicationManager** and other subscribed clients. **FaultAnalyzer** receives all the fault reports from the **FaultNotifier** and generates condensed fault reports.

*Recovery Management* is implemented by two *mechanisms*, **logging** and **recovery**, exploiting two IDL interfaces (**Checkpointable** and **Updateable**) that recoverable application objects have to implement. Objects implementing such interfaces can be invoked by the **logging** and **recovery** mechanisms or by external applications in order to read and write their state. The **logging** mechanism periodically stores on a log information related to an object (state, incoming requests, replies) while the **recovery** mechanism retrieves this information from the log when, for example, spawning a new member to set its initial state.

### 2.2 IRL Architecture

IRL infrastructure is made up of a set of components implemented as CORBA objects. Each request to an object group is transparently handled by IRL, giving the illusion to clients to interact with a single object rather than with an object group.

Each component is deployed in a separate process and is replicated in order not to represent a

single point of failure. Component replicas can be flexibly distributed on distinct FT-CORBA Fault Tolerance Domain<sup>1</sup> hosts, allowing a (static) load balancing of IIOP connections and more effective fault tolerance.

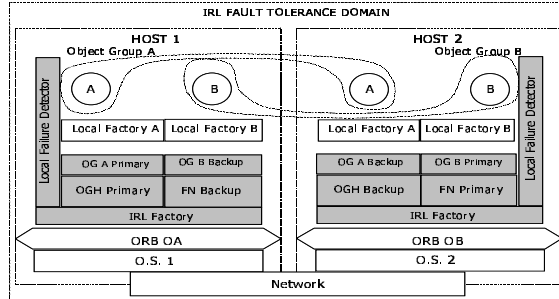


Figure 1: IRL Architecture

In Figure 1 a simple IRL-based system is shown. IRL components are depicted as gray boxes, while non-IRL components (e.g. object group members, local factories, ORBs etc.) are white boxes. IRL components cooperate in order to meet the FT-CORBA specification requirements as follows:

**Replication Management.** This functionality is implemented by the cooperation among the Object Group Handler component (OGH) and a set of Object Group components (OGs). When creating a new object group, members are instantiated by such components using *Local Factories*<sup>2</sup>.

**Fault Management.** This functionality is implemented by the cooperation among the Local Failure Detectors (LFDs) and the Fault Notifier (FN) components. In particular, LFDs are object-level FT-CORBA compliant failure detectors pushing fault reports to FN.

**Recovery Management.** Recoverable replicated objects have to implement the *Checkpointable* interface. Recovery is carried out by the IRL OG component that exploits local factories to spawn new replicas and the *Checkpointable* methods to perform state synchronization.

In addition to implementing a subset of the FT-CORBA specification, IRL exhibit the interoperability property, i.e., object replicas and IRL components can be deployed on ORBs from distinct vendors. This is achieved as IRL components cooperate using standard CORBA invocations.

In the following, a detailed description of each IRL component role is given.

**Object Group (OG).** An Object Group object is associated to each group of replicated application CORBA servers. OG stores the IORs of the members, the information about their availability and the FTPs of the group. OG is also responsible for enforcing strong replica consistency within its group. In particular, OG receives all the requests directed to its object group and imposes a total order on them (it actually acts as a request serializer).

**Object Group Handler (OGH).** One Object Group Handler object is associated to a FTD. This component exports the FT-CORBA *ReplicationManager* interface and it is responsible for creating new object groups and publishing their IOGR.

**Local Failure Detector (LFD).** LFDs are FT-CORBA compliant **FailureDetectors**. A distinct LFD component is deployed on each FTD host and is responsible for detecting failures of all the monitorable CORBA objects running on its own host. These objects are both IRL components (i.e. IRLF, OGH, OG and FN) and object group members. OGH and IRLF can request LFD to monitor a specific CORBA object. LFD maintains a list whose entries are IORs of the monitorable objects residing on its host, along with their failure detection FTPs. LFD monitors such objects by periodically invoking their FT-CORBA *is\_alive()* method. If an object does not reply within the timeout value defined in its FTPs, LFD pushes a fault report to FN. Moreover,

<sup>1</sup> The FT-CORBA specification introduces *fault tolerance domains* (FTDs) to allow application scalability. A FTD is composed by several interconnected CORBA hosts housing members of possibly distinct object groups. Each replica of a same object group has to be deployed on a distinct host belonging to a single FTD. Distinct FTDs can be interconnected by *inter-domain gateways*.

<sup>2</sup> Local factories are implemented by the application developers and a distinct local factory has to be developed for each distinct object type and deployed on each FTD host.

each LFD notifies the liveness of the host it resides on by sending heartbeats to FN.

**Fault Notifier (FN).** The Fault Notifier is a FT-CORBA compliant **FaultNotifier**. It receives fault reports from LFDs and subscriptions for failure notifications by any interested client object. When FN receives a failure notification from a LFD, it forwards this notification to every interested client. In addition to this, FN implements host fault monitoring by receiving heartbeats from LFD (see Section 3.1).

**IRL Factory (IRLF).** A distinct IRLF component runs on each FTD host and exports interfaces for creating IRL components at infrastructure start-up or after a component crash. In particular each IRLF can instantiate on its host new LFD, OGH, OG and FN components. IRLF has to be installed on every ORB of a given FTD.

### 3 Failure Management of IRL Components

A mandatory FT-CORBA requirement is the absence of single points of failure in a compliant FTI therefore each IRL component must be fault-tolerant. In particular, we decided that each IRL component has to be replicated using a passive replication technique. The choice of the appropriate passive replication technique (e.g. *stateless, cold, hot*) of each IRL component depends on its deployment and fault tolerance requirements. In particular, we can identify two classes of components, Host-Specific IRL Components (LFD and IRLF) and Domain-Specific IRL Components (OG, OGH and FN).

#### 3.1 Host-Specific IRL Components

These components are installed on each FTD host and their activities are related to their host (i.e., LFD monitors objects running on *its* host, IRLF creates objects on *its* host). As a consequence, host-specific components do not need to survive to their host crash. However, being prone to software failures they are replicated and their replication is efficiently performed on a local basis (i.e. with no message exchange between different hosts).

**IRL Factories.** Being IRLF a stateless component, we adopted a stateless primary-single-backup replication style: on each host are deployed a primary and a backup IRLF. The failure detection method is based on bi-directional flow of heartbeats (as they are on the same host, heartbeats can be implemented by using, for example, IPC). When one entity considers the other one as crashed (no heartbeat received for a certain amount of time) it recreates the partner. If the primary has crashed, the backup, as first action, takes over the primary role.

**Local Failure Detectors.** LFD has a state composed by the list of monitorable objects and their fault-monitoring properties therefore we adopted a cold passive replication style. Each time LFD receives references and properties of a new object to be monitored, it stores them on a log. If LFD crashes, IRLF creates a new instance that sets its initial state to the last state stored by the crashed LFD. The event of a LFD crash is monitored by IRLF by setting a timeout each time LFD invokes IRLF `is_alive()` method. Furthermore, as mentioned in Section 2.1, LFD periodically sends heartbeats to FN (push-style host failure detection). When a LFD crashes, FN stops receiving its heartbeats and could detect an host failure. Due to the replication style used by LFD, FN has to wait a sufficient amount of time before declaring the host crashed. This amount has to take into account the network round-trip time and the potential software failure detection and recovery delays of LFD in the remote host. Figure 2(a) illustrates the messages exchanged among host-specific components and their relationship.

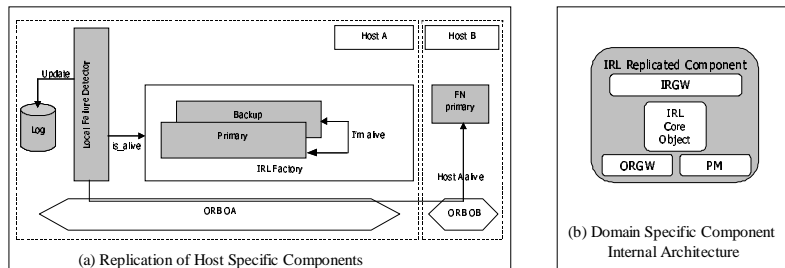


Figure 2: HSC Replication and DSC Internal Architecture

### 3.2 Domain-Specific IRL Components

IRL domain-specific components (DSCs), such as OGH, OG and FN, are replicated following a hot-passive (primary-backup) replication style, allowing fast recovery. The primary receives all the requests directed to its group. When a request modifies its state, the DSC primary updates its backup(s) in order to maintain strong replica consistency upon recovery. Furthermore, while serving a request, a primary DSC could perform outgoing requests (e.g., OGH and OGs cooperate in order to implement the **ReplicationManager** functionality). In such a case strong replica consistency upon recovery is achieved by the primary (i) identifying outgoing requests to avoid repeated executions and (ii) notifying the backups *before starting* and *at the end of* each outgoing request invocation. *Transparent primary failover* of a DSC component includes *failure notification*, *election of a new primary* and *consistent recovery*. To handle these mechanisms, each IRL DSC contains a set of CORBA objects running in a single addressable space, i.e. the PersistenceManager, the OutgoingRequestGateway and the IncomingRequestGateway (Figure 2(b)), that surround the IRL Core Object (the one that actually implements the failure-free behavior of the DSC object in primary configuration, described in Section 2.2).

In particular, the PersistenceManager role is updating the state of the backups and, in case of a primary crash, starting an election protocol to elect a new primary.

Updates are executed through a set of successive CORBA one-way invocations identified by a sequence number. If a primary crashes during an update, the new primary is chosen among the most updated backups. Then, the new primary PersistenceManager sends the last received update to all the backups, to ensure they all are synchronized. If the primary failure occurred during the execution of an outgoing request, the request will be executed again by the new primary. The OutgoingRequestGateway of a DSC adds a tag to each request to uniquely identify it, so that the IncomingRequestGateway of a DSC can filter out repeated requests.

## 4 CORBA Client-Server Interactions in IRL

The interaction of a client with an object group is mediated by the OG IRL component associated to the group. Requests are received by the primary OG that relays them to each group member. The address of the primary OG is contained in the IOGR, which was published by OGH during the creation phase of the group. The IOGR of a given object group is composed by the IORs of the OG replicas (*not* of the replicated servers).

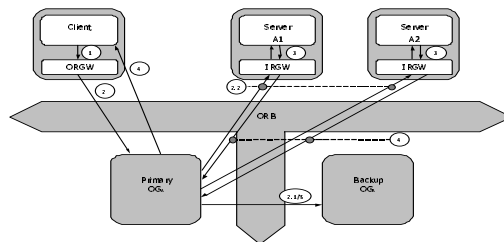


Figure 3: A Fault-free Client-Server Interaction Mediated by IRL

If running on a CORBA 2.x compliant ORBs, IRL clients and servers have no support neither for transparent client reinvoication/redirection nor for repeated executions of re-invoked requests. To overcome these limitations, corba 2.x IRL clients are equipped with a *portable client interceptor* embedding an ORGW IRL object and servers with a *portable server interceptor* embedding an IRGW IRL Object ([11]). If the primary OG fails during the processing of a request, the client interceptor transparently re-invokes the request on a backup OG, using the alternate IORs contained in the IOGR.

Figure 3 illustrates the sequence of invocations performed to complete a client-server interaction in a fault-free scenario. The steps are:

1. The client invokes the remote method. This is captured by its portable interceptor and processed by the client ORGW that augments it with a unique service context and starts invoking the primary OG.

2. The request reaches the primary OG.
  - 2.1 The PM of the primary OG updates the state of its backup notifying it that an outgoing request is in progress.
  - 2.2 The primary OG forwards the request to the members of the group by means of a set of *concurrent* remote CORBA invocations, one for each group member. The primary OG actually acts as a serializer of the requests within server replicas group.
3. The request is intercepted by the *server portable interceptor* and processed by IRGW that checks if it has not been already processed. In the affirmative, IRGW relays the request to the server and, then, stores the result and sends it to the primary OG. Otherwise, IRGW uses the stored result as the reply to be sent to primary OG.
4. When the primary OG receives *the first* reply from the group, it returns the result to the client.
5. When all the replies from non-crashed group members are arrived (if a group member crashes, it will be detected by IRL fault management system), the PersistenceManager of the primary OG updates the state of the backup.

## 5 Conclusions and Future Work

In this paper we presented IRL, an interoperable FT-CORBA compliant platform whose implementation is currently in progress in our department. IRL provides CORBA clients with transparent interactions with actively replicated CORBA servers and transparent, consistent server failovers. To get interoperability, IRL components are all deployed above the ORBs. As a consequence, there is a performance degradation of the failure-free end-to-end client-server interaction due to the use of standard CORBA invocations for the cooperation among the entities. Currently, this suggests to trade a certain (small) probability of a service downtime with higher performance. Therefore, we are investigating, for each IRL component, the best replication protocol and replication degree in order to lower client-server interaction latency while maximizing the platform reliability.

## References

- [1] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz, *AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects*, Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (West Lafayette, IN, USA), October 1998, pp. 245–253.
- [2] P. Felber, *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*, Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998, PhD thesis no. 1867.
- [3] P. A. Felber and R. Guerraoui, *The Implementation of a CORBA Group Communication Service*, Theory and Practice of Object Systems 4 (1998), no. 2, 93–105.
- [4] R. Guerraoui and A. Shiper, *Software-based replication for fault tolerance*, IEEE Computer - Special Issue on Fault Tolerance 30 (1997), 68–74.
- [5] S. Landis and S. Maffei, *Building Reliable Distributed Systems with CORBA*, Theory and Practice of Object Systems 3 (1997), no. 1.
- [6] C. Marchetti, M. Mecella, A. Virgillito, and R. Baldoni, *An Interoperable Replication Logic for CORBA Systems*, Proceedings of the 2nd International Symposium on Distributed Objects and Applications (Antwerpen, Belgium), September 2000, pp. 7–16.
- [7] C. Marchetti, A. Virgillito, R. Baldoni, and M. Mecella, *Integrating Autonomous Enterprise Systems through Dependable CORBA Objects*, Proceedings of the 5th International Symposium on Autonomous Decentralized Systems ISADS (ISADS01) (Dallas, Texas, USA), March 2001, pp. 204–211.
- [8] L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, L.A. Tewksbury, and V. Kalogeraki, *The Eternal System: an Architecture for Enterprise Applications*, Proceedings of the 3rd International Enterprise Distributed Object Computing Conference (EDOC'99) (Mannheim, Germany), July 1999, pp. 214–222.
- [9] B. Natarajan, A. Gokhale, S. Yajnik, and D.C. Schmidt, *DOORS: Towards High-Performance Fault Tolerant CORBA*, Proceedings of the 2nd International Symposium on Distributed Objects and Applications (Antwerpen, Belgium), September 2000, pp. 39–48.
- [10] Object Management Group (OMG), Framingham, MA, USA, *Fault Tolerant CORBA Using Entity Redundancy - Request For Proposal*, OMG TC document ed., December 1998.
- [11] Object Management Group (OMG), Framingham, MA, USA, *Portable Interceptor Specification*, OMG Document orbos ed., December 1999, OMG Final Adopted Specification.
- [12] Object Management Group (OMG), Framingham, MA, USA, *The Common Object Request Broker Architecture and Specifications. Revision 2.4.1*, OMG Document formal ed., November 2000, OMG Final Adopted Specification.
- [13] Object Management Group (OMG), Framingham, MA, USA, *Fault Tolerant CORBA Specification, V1.0*, OMG Document formal ed., April 2000, OMG Final Adopted Specification.