

Full Paper

Enhancing Availability of Cooperative Applications through Interoperable Middleware

CARLO MARCHETTI, ANTONINO VIRGILLITO AND ROBERTO BALDONI

Dipartimento di Informatica e Sistemistica

Università "La Sapienza" di Roma

Via Salaria 113, 00198, Roma, Italy.

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

Cooperative information systems are characterized by distribution, high heterogeneity and scale. Therefore they require interoperable, dependable services on top of which the development of cooperative application can take place. This paper studies, in the context of the Unitary Network of the Italian Public Administration, the problem of increasing the availability of the services exported by a cooperating entity through interoperable middleware. In particular we show how a Fault Tolerant CORBA compliant system, namely the Interoperable Replication Logic (IRL), can be used to increase such availability by building a replicated cooperative gateway that wraps enterprise cooperative applications.

Keywords: Cooperative Information Systems, Fault Tolerance, Object Replication, CORBA, FT-CORBA, Middleware Platforms

1 INTRODUCTION

A *cooperative information system* (CIS) is a collection of cooperating, distributed information systems interconnected by a wide and complex communication network ([1], [2]). CISs are characterized by high heterogeneity in terms of platforms and computing environments that make difficult the cooperation among remote entities. However, the CIS model can be applied in many "e-contexts", such as *e-government*, *e-business* etc. and different proposals ([3], [4]) are trying to address the problem of developing services for cooperative applications in such system model.

The *Unitary Network of the Public Administration* (RUPA¹) is a challenging project promoted by the Italian Government to allow the cooperation at the application-level of the large set of autonomous and heterogeneous information systems of the Italian Public Administration, which in the past followed an independent and uncoordinated design and

¹ The acronym of RUPA comes from the translation in Italian of "Public Administration Unitary Network", i.e. "Rete Unitaria della Pubblica Amministrazione" (RUPA).

development [5]. In RUPA, the heterogeneity of these systems has been managed by a *cooperative architecture* [6] that uses *application gateways* to standardize the access to heterogeneous resources contained in a *RUPA domain*. A domain is the RUPA abstraction of an information system belonging to a single organization while gateways are object-oriented wrappers of the domain systems providing services according to a predefined interface.

Gateways being the only access point for all the services exported by a RUPA domain, their fault tolerance and high availability represent key issues in order to guarantee quality of service (QoS) to RUPA clients. Fault-tolerance and high availability can be obtained by software replication: a service can survive a fault if it is provided by a set of server replicas hosted by distinct servers. 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 (state machine approach) or passive replication (primary-backup approach) [7].

The object-oriented wrapping implemented by RUPA domain gateways can be easily implemented exploiting a distributed object-oriented middleware such as the Common Object Request Broker Architecture (CORBA [8], [9]). CORBA is a standard middleware on top of which Distributed Object Computing (DOC) applications can be designed, implemented and deployed shielding developers from tedious and error-prone aspects. Furthermore, CORBA presents high object oriented wrapping capabilities that fit well in cooperative heterogeneous environments. However, until revision 2.4, 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 [10], OGS [11][12], DOORS [13][14][15], Isis+Orbix [16], Electra [16], AQuA [17], IRL [18].
- The Object Management Group (OMG) issued a RFP in 1998 that produced, in early 2000, the Fault Tolerant CORBA specification [19]. 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 that allows to manage consistently replicated, fault-tolerant CORBA objects. FT-CORBA standard has been included in the CORBA 3.0 specification.

IRL (Interoperable Replication Logic) is a software infrastructure that can provide fault tolerance and high availability to applications running over different CORBA implementations (Interoperability property). After the approval of the FT-CORBA standard, we started a re-engineering process on the first IRL design ([18]) that resulted in a new design that adopts standard compliant interfaces and allows a more flexible deployment.

In this paper we present the new FT-CORBA compliant IRL design and analyze its use in the context of RUPA cooperative architecture in order to increase the availability of the application gateways by exploiting IRL interoperability property. Increasing the gateway availability with IRL means increasing the availability of the whole integration layer, allowing each domain to provide its own cooperative services and applications trusting the availability of the other domains.

The remainder of this paper is organized as follows: Section 2 illustrates RUPA and its availability requirements, Section 3 outlines IRL system, Section 4 describes how to increase RUPA service availability by using IRL system and Section 5 presents a comparison of fault tolerance CORBA systems with respect to the problem of cooperation and integration in large-scale systems. Section 6 concludes the paper.

2 THE UNITARY NETWORK OF THE ITALIAN PUBLIC ADMINISTRATION (RUPA)

The Unitary Network of the Italian Public Administration (RUPA) is the most important and challenging project undertaken by the “Autorità per l’Informatica nella Pubblica Amministrazione” (AIPA – Authority for IT in the Public Administration, [20]) since the beginning of its mandate, given by the Italian Government in 1993. RUPA final objective is the implementation of a “secure Intranet” interconnecting the information systems of different Italian public administrations (e.g. departments, ministries and organizations). The emphasis of the project is on promoting cooperation among the various administrations at the *application level*. Besides providing the essential interconnection services (e-mail, file transfer, etc.) to administrations, the project defines the Unitary Information System of the Italian Public Administration as a whole, by bringing together the collection of distributed, autonomous enterprise systems of each administration into a common cooperative architecture. In turn, this will allow to reengineer global administrative processes by making more effective use of the information made available by each individual system.

2.1 RUPA Architecture

As depicted in Figure 1, RUPA architecture consists of three functional layers, i.e. the *Transport*, *Interoperability* and *Cooperative Service* layers. The Cooperative Service layer includes the middleware services that enable the development and the deployment of new inter-administration cooperative applications. The distributed computing model based upon the Cooperative Service layer is called *cooperative architecture*.

A RUPA fundamental concept is the *domain*: a RUPA domain includes all the computing resources, networks, applications and data that belong to a *specific* administration, regardless of the technical nature of such information systems.

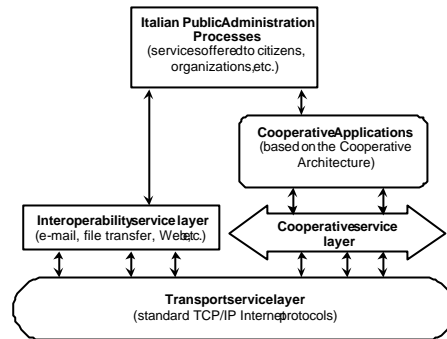


Fig. 1. RUPA Service Layers

In practice, the computing environment of a domain often includes mainframe-based legacy applications (as in the case of the Italian Social Security Service – ISSS, the Ministry of Justice – MoJ, or the Ministry of Finance – MoF) as well as high-end and departmental computing systems, even spread over a wide geographical area. In other situations (as in many City Councils – CCs, or in many Prefectures – PrefFs) it must be developed from scratch (there is no information infrastructure other than simple office automation applications). Therefore a RUPA domain is, in the most general case, a distributed information system. The whole RUPA is a *cooperative* information system.

Once it is connected to RUPA, each domain is modeled as a single entity, regardless of its internal complexity and geographical extension. The domain functionality is made available through a *domain cooperative gateway* (DC gateway) that implements the set of services described in the export interfaces of the domain.² Figure 2 shows the Unitary Network architecture. In this figure, $S_1 \dots S_n$ are heterogeneous MoF information systems, interconnected by a nation-wide area intranet.

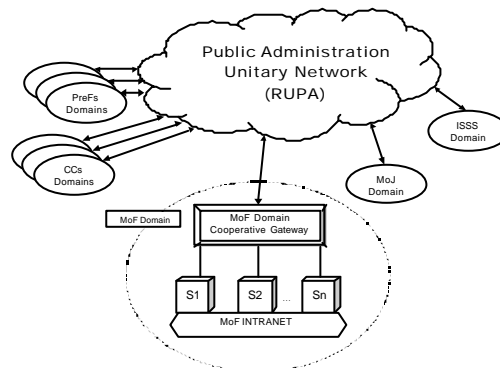


Fig. 2. RUPA Cooperative Architecture

² Note that a DC Gateway can be either *centralized*, i.e. composed by one or a few hosts deployed over a LAN, or even *distributed* over a geographical area.

2.2 Inter-Domain Cooperation in RUPA

Inter-Domain cooperation in RUPA follows the cooperative architecture model. Interfaces of the services that a domain exports can be specified according to the DOC approach: first, a collection of business objects and components is specified using an appropriate *Interface Definition Language*, then implemented in DC gateways, and finally made available to other domains by deploying the gateways as object/component servers [21] (e.g. CORBA server objects). In order to make accessible the services it offers, each domain is required to define all the interfaces to be exposed and to implement them. End users and applications belonging to a domain interacts with DC gateways by means of a local *Domain Proxy Gateway* (DP gateway) that has knowledge about the services available in RUPA and about the protocols and interfaces required to access them (Figure 3).

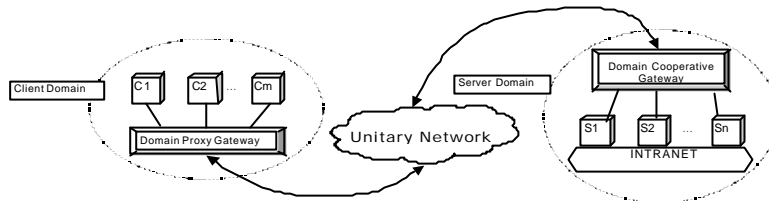


Fig. 3. Domain Cooperative (DC) and Domain Proxy (DP) Gateways in RUPA

Note that a domain can behave both as a client and as server: in this case it must implement both gateways.

DC gateways actually become the mid-tier of a layered architecture and carry out the basic function of encapsulating existing back-office systems in object-oriented wrappers that are seen as business objects from RUPA clients point of view.

2.3 The need for Availability in RUPA

AIPA specifications about the QoS of the Unitary Network ([5], [22], [6]) states that services offered by a particular domain must be highly available. In particular their overall availability must be higher than 99,8%, in order to allow developers to implement cooperative applications trusting the availability of the systems involved.

This requirement can be satisfied only by enforcing fault tolerance and high availability policies at the different levels of RUPA architecture, exploiting different techniques:

- **Transport Service Layer Availability.** To maintain connection availability AIPA directives suggest to deploy *redundant backup lines* at the physical level and to adopt *dual-homing routing* techniques at the network layer.
- **Domain Services Availability.** In RUPA, the availability of a given service depends on the availability of the server domain systems (Server Availability) as well as of the DC gateway wrapping them (DC Gateway Availability):
 - **Server Availability.** In order to face the problem of adding fault tolerance and high availability to a particular domain system, it can be used common techniques and practices as well as available products, that need to be customized in order to face the particular system.

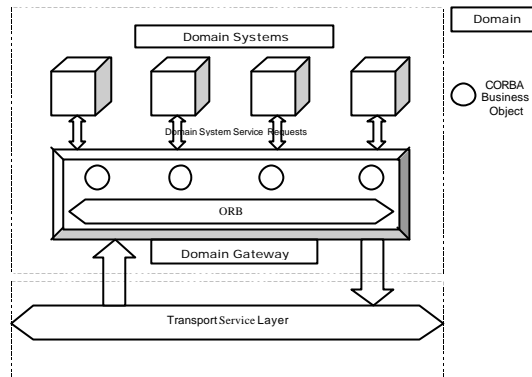


Fig. 4. A RUPA DC gateway as Business Object Container

- DC Gateway Availability.** As pointed out in the previous section, DC gateways are business object containers. A business object is the object oriented wrapping of a particular domain service provided by one or more systems belonging to a domain. In Figure 4 is represented a domain and its gateway as a business object container³. RUPA clients perceive a fault of a particular DC gateway business object as the fault of the entire domain system it wraps, even if this system is still up and ready to accept service requests. Moreover, while a fault on a domain system can result in the loss of the service it offers, a fault of the gateway itself can result in the loss of availability of all the services exposed by the domain. Therefore a DC gateway, or better all the business objects it hosts, are mission critical objects from RUPA QoS point of view and particular care should be taken in their design with respect to fault tolerance and high availability issues.

3 IRL – INTEROPERABLE REPLICATION LOGIC

In a previous work [18], we presented a first version of an interoperable CORBA-compliant middleware, namely IRL (Interoperable Replication Logic). IRL guarantees strong replica consistency of CORBA application objects through active replication of application objects themselves and passive replication of IRL architectural components. IRL is based on an “above the ORB” approach, i.e. each IRL component is implemented as a CORBA object and exploits the standard CORBA protocol (IIOP) for communications. This allows deploying IRL on top of CORBA implementations from different vendors (interoperability property), overcoming one well-known limitation of the FT-CORBA specification (i.e. the “vendor-dependency” limitation, [19]).

However, the first IRL design lacked FT-CORBA compliance and suffered for poor performance. The new IRL design is the result of a full re-engineering process during which all the component interfaces have been adapted to the FT-CORBA standard and

³ For simplicity, in Figure 4 we assumed a one-to-one mapping between business objects and domain systems, i.e. each business object executes transactions on one domain system and all of the transactions of a domain system are wrapped by exactly one gateway business object.

the whole architecture has been made more flexible, reducing performance bottlenecks and allowing an easy and effective deployment over heterogeneous environments.

In this section we first present a brief overview of the FT-CORBA specification, then we introduce the new design of IRL architecture, describing all the components, their relationship with the FT-CORBA standard and the failure management mechanism that avoid that IRL components are single points of failure. Finally, an example of a client-server interaction showing the behaviour of the architecture is given.

3.1 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 *object group* abstraction ([23]) 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, [8]) 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). FT-CORBA compliant client ORBs uses IOGRs 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 main features: *replication management*, *fault management* and *recovery management*.

Replication management consists in the creation/removal of object groups or object group members and in FTPs 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. When a **FaultDetector** assume a crash of a replica, it pushes 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, i.e. *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 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 consistently.

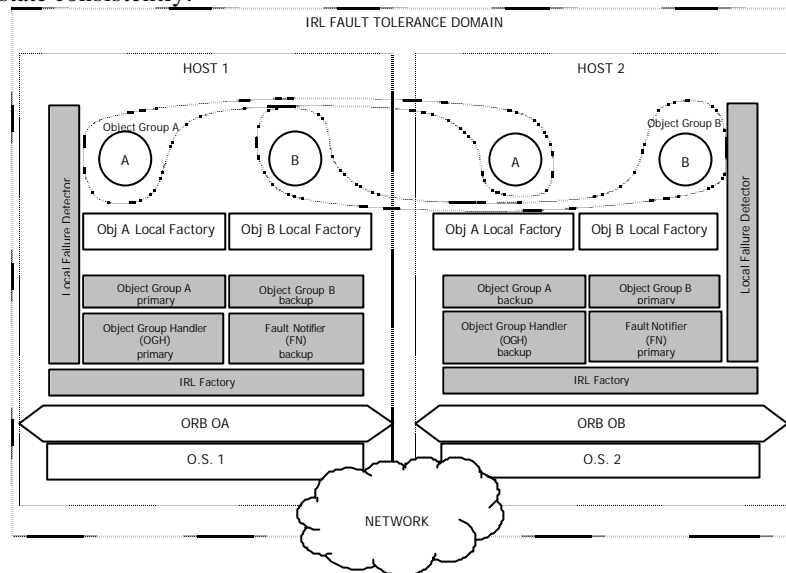


Fig. 5. IRL Architecture

3.2 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. In this way, component replicas can be flexibly distributed on distinct hosts belonging to a given FT-CORBA Fault Tolerance Domain⁴, allowing a (static) load balancing of IIOP connections and more effective fault tolerance.

In Figure 5 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 between the Object Group Handler component (OGH) and a set of Object Group components (OGs). When creating a new object group, members are instantiated by

⁴ 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*.

such components using *Local Factories*⁵.

- *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 the following, a short description of each IRL component role is given.

Object Group (OG). One instance of 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 instance of an Object Group Handler object is associated to each FTD. This component exports the FT-CORBA *ReplicationManager* interface and it is responsible for creating new object groups and publishing their IOGRs.

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

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 3.3.1).

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 host of a given FTD.

3.3 Failure Management of IRL Components

A mandatory FT-CORBA requirement is the absence of single points of failure in a

⁵ The application developers implement local factories and a distinct local factory has to be developed for each distinct object type and deployed on each FTD host.

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).

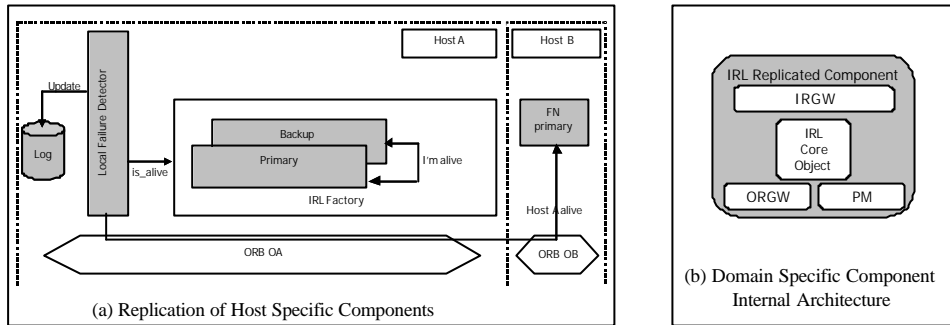


Fig. 6. HSC Replication (a) and DSC Internal Architecture (b)

3.3.1 Host-Specific IRL Components

These components are installed on each FTI 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 their host crash. However, they are replicated and their replication is efficiently performed on a local basis (i.e. with no message exchanges between different hosts) to survive replica crashes.

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 maintains 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 3.2, LFD periodically sends heartbeats to FN (push-style host failure detection). When a LFD crashes, FN stops receiving its

heartbeats and could detect a host failure⁶. Figure 6(a) illustrates the messages exchanged among host-specific components and their relationship.

3.3.2 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 addressed 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 the same addressable space, i.e. the PersistenceManager, the OutgoingRequestGateway and the IncomingRequestGateway (Figure 6(b)) that logically surround the IRL Core Object (the one that actually implements the failure-free behavior of the DSC object in primary configuration, described in Section 3.2).

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.

3.4 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 is 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).

If running on a CORBA 2.x compliant ORBs, IRL clients and servers have no support neither for transparent client reinvocation/redirection nor for repeated executions of re-invoked requests. To overcome these limitations, CORBA 2.x IRL clients and servers are enhanced respectively with a *portable client interceptor* ([24]) embedding an ORGW IRL object and with a *portable server interceptor* embedding an IRGW IRL Object. If the

⁶ 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 local failure detection and recovery delays of LFD in the remote host.

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. Currently, the IRL prototype implements a replication protocol based on simplifying assumption on the underlying system model. The protocol is described in the following and detailed in [25]⁷. Figure 7 illustrates the protocol, highlighting 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 request identifier 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 replica 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.

⁷ The replication protocol can be easily modified to face different application requirements in terms of consistency and performance. An enhanced version of the protocol, currently being implemented, is described in [26].

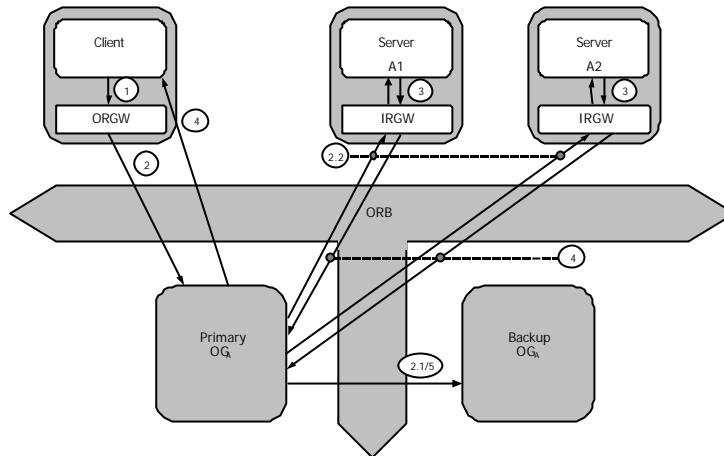


Fig. 7. A Fault-free Client-Server Interaction Mediated by IRL

4 IRL-BASED HIGHLY-AVAILABLE DC GATEWAYS

In this section we show how IRL can be applied in the RUPA context to increase the availability of DC gateways.

4.1 IRL in the Autonomous Systems of the Unitary Network

In order to preserve the autonomy of the administrations, AIPA has neither fixed a particular platform nor a specific technology on which deploy DC gateways. IRL interoperability allows the deployment of the architecture itself as well as of application object replicas on different ORBs running on different platform. Moreover, IRL can be installed without requiring any system upgrade and on top of different ORBs⁸. Deploying IRL on the domain gateways allows the replication of the business objects, making them fault tolerant and highly available. Replicating all the business object of a particular DC gateway actually means replicating the whole gateway and increasing the availability of all the domain services, by decoupling the service availability from the DC gateway availability. Figure 8 illustrates an IRL-based DC gateway.

⁸ We have developed an IRL prototype running on three Java ORBs (JacORB 25, Orbus 28 and Orbix 2000 for Java 29).

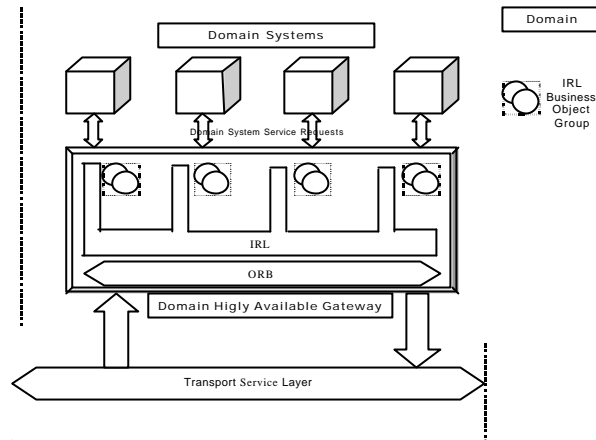


Fig. 8. A Highly-Available DC gateway

4.2 IRL DC Gateway Deployment

Figure 9 shows a possible internal configuration of an IRL-based highly available DC gateway housing two business object groups, A and B. Local factories and IRL host specific components are shown in white, primary IRL components are shown in light gray and backup IRL components are shown in dark gray. To avoid the presence of a (*hardware*) single point of failure, business objects as well as IRL components are replicated on three hosts (Host1, Host2 and Host3). Business objects are replicated using a passive replication technique. Passive replication allows to distribute the two primary replicas on distinct hosts, sharing the computational load⁹ between them.

In the configuration shown in Figure 9, we used the same hosts to deploy both IRL components and business object group members. This is actually not necessary in practice: business objects can run on hosts different from the ones on which infrastructure component have been installed.

⁹ Active replication does not allow such a static load sharing. Moreover, in the case of active replication, the domain service requests outgoing from the business objects are replicated.

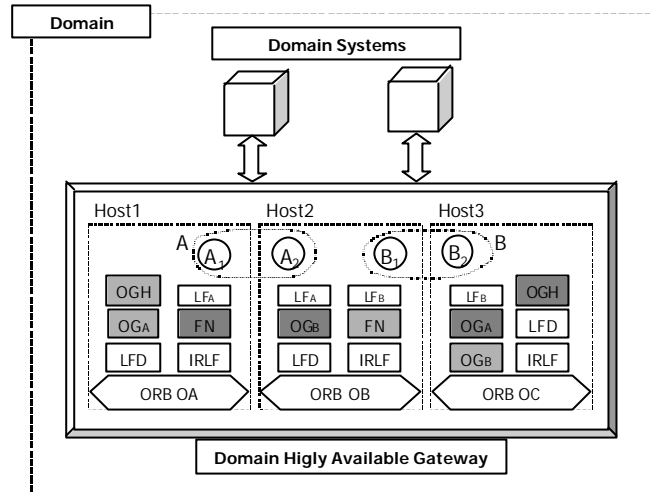


Fig. 9. IRL-Based Highly-Available DC gateway Deployment

5 RELATED WORK

While fault tolerance in cooperative information systems is an emergent topic, a lot of work has been done in the last years to add fault tolerance to CORBA applications. As a result, different systems providing fault tolerance to CORBA application have been proposed in the literature.

A first classification of these systems was proposed by Felber et al. in [12]. In this paper, systems are classified according to the different techniques and approaches (*integration, interception, service*) they exploit to provide fault tolerance. A different taxonomy, that compares systems from an architectural point of view, can be found in [18]. This classification compares systems according to the degree of *intrusion* between their replication logic and the standard CORBA ORB mechanisms.

In order to investigate the fitness degree of each system when facing heterogeneity and dimension of the autonomous enterprise systems belonging to a CIS, we classify them in two categories: *Group Toolkit based* and *IIOF based*.

5.3 Group Toolkit based Fault Tolerance CORBA Systems

A group toolkit is a proprietary communication and monitoring system that can be used to handle software replication [23]. It offers services such as group membership management and reliable, ordered group communication. Group toolkit based systems relay the task of managing group membership and communications to a group toolkit (i.e. the replication logic, or a part of it, is logically settled “below” the ORB). Systems that fall under this category are described below:

- The Isis+Orbix system is an intrusive extension of the Orbix ORB ([29]) that has

been modified in order to add support for object replication by exploiting a group toolkit (namely, ISIS, [30]).

- In the Eternal system ([10]), replication is accomplished by interposing a particular layer, the interception layer, between a generic ORB and the OS. This layer, that is an OS dependant object allowing non-intrusive design with respect to the ORB, captures CORBA IIOP messages and “passes” them to a group toolkit (Totem, [31]). Eternal is FT-CORBA compliant.
- Also AQuA exploits a group toolkit (Maestro/Ensemble, [2], [3]) to maintain consistency among replicated CORBA objects, but differs from the Eternal system mainly for the way the requests are passed to it. In particular, in AQuA the IIOP module of a particular ORB is intrusively replaced by a gateway that passes requests to the group toolkit.

5.4 IIOP based Fault Tolerance CORBA Systems

In the IIOP based systems, the replication logic is logically settled “above” the ORB. Therefore the task of managing replica consistency is all done by CORBA compliant objects through IIOP messages. IRL is an IIOP based system. Other systems that falls under this category are described below:

- In the Object Group Services (OGS) the functions commonly available in a group toolkit have been implemented as a CORBA Service (COS), i.e. as a set of IDL interfaces that can be explicitly used by application objects to handle their replication.
- DOORS is a FT-CORBA compliant system that, differently from OGS and IRL, implements only passive replication and application controlled membership but does not guarantee strong replica consistency, i.e. this is left to the application developers.

5.5 Discussion

Group toolkit based systems seem not to fit well when facing heterogeneity and scale of CISs. Group toolkits are proprietary tools, often specialized to a restricted set of platforms, and thus they suffer the intrinsic heterogeneity of CISs. Moreover, both the geographical distribution of CISs and the large number of hosts, could give rise to scalability issues. At the best of our knowledge, group toolkits have been used only in department (i.e. small scale) local systems. Only a few prototypes ([34], [35]) and some theoretical papers ([36], [37]) address the problem of using a group toolkit over a large-scale distributed system.

On the other hand, IIOP based systems could face heterogeneity exploiting the high wrapping capabilities provided by CORBA. The main drawback resulting from their “above” the ORB design is performance: using only the IIOP protocol excludes the possibility of optimization based on the direct access to the OS functions (as done by group toolkit based systems).

IIOP based systems differs in the mechanism implemented in their replication logic. OGS handles object replication by running *distributed* protocols over IIOP. For this reason, it does not scale well and could exhibit poor performance in large-scale systems, when compared to “centralized” approaches such as the ones proposed by IRL and DOORS. This is due to the higher number of IIOP messages required by its protocols.

DOORS is the system whose design concepts are closer to IRL. It doesn't use group communication protocols, but has a centralized Replica Manager implementing passive replication and fault monitoring. However, DOORS does not handle consistency among the replicated objects, leaving the error-prone task of implementing the replication logic to the application developer.

From this discussion it results that IRL and DOORS¹⁰ seems to be more suitable than other systems to be used for enhancing cooperative application with high availability. However, in small scale system such a *centralized* DC gateway and domain systems, group toolkit based systems could also be used, probably resulting in better performance than IRL, because of their access to lower-level network protocols (TCP/IP, UDP multicast etc.).

6 CONCLUSIONS

In this paper we have shown how the cooperative applications of the heterogeneous, autonomous enterprise system of the Italian Public Administration can be integrated in the Unitary Network (RUPA) through CORBA dependable objects, deployed on DC gateways and managed by the Interoperable Replication Logic (IRL) system. This "dependable" integration should allow each Administration to develop its own highly available cooperative applications trusting the availability of the business object deployed on the DC gateways of the other domains. IRL design has been also briefly described along with a discussion about the motivations that should promote it in the CIS environment. However, in small scale and local system, group toolkit based solutions could effectively and efficiently accomplish the task of dependable integration platform.

7 ACKNOWLEDGEMENTS

The authors would like to thank Massimo Mecella, for the relevant suggestions and interesting discussions, and Raffaella Panella, Paolo Papa, Alessandro Termini, Sara Tucci Piergiovanni and Luigi Verde, for their valuable contributions to the IRL project. This work was partially supported by AMS and by the MIUR, COFIN 2001 Project "DaQuinCIS – Methodologies and Tools for Data Quality inside Cooperative Information Systems". A preliminary version of this paper has been presented in the IEEE International Symposium on Autonomous Decentralized Systems (ISADS '01), Dallas, 2001.

¹⁰ In its most recent version the DOORS design team has adopted the TAO ORB [8] as underlying ORB platform. In our opinion, though this design choice provides a performance improvement, at the same time it limits the interoperability of the DOORS platform.

REFERENCES

1. M. L. Brodie, "The Cooperative Computing Initiative. A Contribution to the Middleware and Software Technologies", GTE Laboratories Technical Publication, 1998, <http://info.gte.com/pubs/PITAC3.pdf>
2. J. Mylopoulos, M. Papazoglou (eds), "Cooperative Information Systems", *IEEE Expert Intelligent Systems & Their Applications*, vol. 12, no. 5, 1997.
3. A.K. Elmagarmid, W.J. McIver Jr. (eds.): "Special Issue on Digital Government". *IEEE Computer*, vol. 34, no. 2, February 2001.
4. F. Casati, D. Georgakopoulos et al. (eds.), *First Workshop on Technology for E-Services*, Cairo, Egypt, September 14-15, 2000.
5. AIPA: "Public Administration's Unified Network". Roma, Gennaio 1996. [http://www.aipa.it/english\[4/unifiednetwork\[2/feasibilitystudy\[1/](http://www.aipa.it/english[4/unifiednetwork[2/feasibilitystudy[1/)
6. AIPA: "Studio di Prefattibilità per l'Individuazione delle Infrastrutture Necessarie a Favorire lo Sviluppo della Cooperazione Applicativa tra Amministrazioni". Roma, Ottobre 1998. (in Italian)
7. R. Guerraoui, A. Schiper, "Software-Based Replication for Fault Tolerance", in A.K. Somani, N.H. Vaidya (eds.), *Special Section on Fault Tolerance*, IEEE Computer, April 1997.
8. Object Management Group (OMG), *The Common Object Request Broker Architecture and Specifications*. Revision 2.4.1, OMG Final Adopted Specification, OMG Document formal/2000-11-07ed., OMG, Framingham, MA, 2000.
9. R. Soley (ed.), *Object Management Architecture (OMA) Guide*. Second Revision. OMG Framingham, MA, 1992.
10. 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.
11. 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.
12. 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.
13. 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.
14. Andy Gokhale, Bala Natarajan, Douglas C. Schmidt and Shalini Yajnik, "Applying Patterns to Improve the Performance of Fault-Tolerant CORBA", in *Proceedings of the 7th International Conference on High Performance Computing (HiPC 2000)*, ACM/IEEE, Bangalore, India, December 2000.
15. Andy Gokhale, Bala Natarajan, Douglas C. Schmidt and Shalini Yajnik, "DOORS: Towards High-performance Fault-Tolerant CORBA", in *Proceedings of the 2nd International Symposium on Distributed Objects and Applications (DOA '00)*, OMG, Antwerp, Belgium, September 2000.

16. S. Landis, S. Maffei, "Building Reliable Distributed Systems with CORBA", *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.
17. 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.
18. C. Marchetti, M. Mecella, A. Virgillito, R. Baldoni, "An Interoperable Replication Logic for CORBA Systems", in *Proceeding of the 2nd International Symposium on Distributed Objects and Applications (DOA '00)*, Antwerp, Belgium, September 2000.
19. Object Management Group (OMG), *Fault Tolerant CORBA Specification, V1.0, OMG Final Adopted Specification*, OMG Document ptc/2000-04-04 ed., OMG, Framingham, MA, 1999.
20. Autorità per l'Informatica nella Pubblica Amministrazione (AIPA): <http://www.aipa.it> .
21. Mecella M., Batini C.: "Cooperation of Heterogeneous Legacy Information Systems: a Methodological Framework". *Proceedings of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000)*, Makuhari, Japan, 2000.
22. AIPA-ASSINFORM: "Architettura Applicativa della Rete Unitaria della Pubblica Amministrazione". Roma, Febbraio 1997 (in Italian).
[http://www.aipa.it/attivita\[2\]/reteunitaria\[1\]/ArchApp\[5\]/indexarch.asp](http://www.aipa.it/attivita[2]/reteunitaria[1]/ArchApp[5]/indexarch.asp) (in Italian)
23. D. Powell (ed.), "Special Section on Group Communication", *Communications of the ACM*, vol. 39, no. 4, pp. 50-97, April 1996.
24. Object Management Group (OMG), *Portable Interceptor Specification, V1.0*, OMG Final Adopted Specification, Document orbos ed., December 1999, OMG Final Adopted Specification.
25. R. Baldoni, C. Marchetti, A. Termini, "IRL: a Three-Tier Approach to FT-CORBA Infrastructures", Technical Report 01-2002, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2002.
26. R. Baldoni, C. Marchetti, S. Tucci Piergiovanni, "Asynchronous Active Replication in Three-Tier Distributed Systems", Technical Report 05-2002, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2002.
27. JacORB web site: <http://jacorb.inf.fu-berlin.de/>
28. ORBacus home page: <http://www.orbacus.com/ob/>
29. Orbix 2000 for Java home page: http://www.iona.com/products/orbix2000_home.htm
30. K. Birman, R. Van Renesse (eds.), *Reliable Distributed Computing with the ISIS toolkit*, IEEE CS Press, 1994.
31. 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.
32. M.G. Hayden, *The Ensemble System*. Ph.D thesis, Cornell University, Ithaca, NY, 1998.
33. A. Vaysburd, K. Birman, "Building Reliable Adaptive Distributed Objects with the Maestro Tools", in *Proceedings of Workshop on Dependable Distributed Object Systems*.
34. Ö. Babaoglu, R. Davoli, L. Giachini, M. Baker, "Relacs: a Communication Infrastructure for Constructing Reliable Applications in Large-scale Distributed Systems", Technical Report UBLCS-94-15, Department of Computer Science, University of Bologna, 1994.
35. C. Malloth, P. Felber, A. Schiper, U. Wilhelm. "Phoenix: a Toolkit for Building Fault-Tolerant Application in Large-Scale", Technical Report, Department d'Informatique, Ecole Polytechnique Federale de Lausanne, 1995.

36. R. Baldoni, R. Friedman, and R. van Renesse, "The Hierarchical Daisy Architecture for Causal Delivery" in *Proceedings of 17th International Conference on Distributed Computing Systems (ICDCS '97)*, Baltimore, MD, 1997.
 37. L. Rodrigues, P. Verissimo, "Causal Separators and Topological Timestamping: an Approach to Support Causal Multicast in Large-Scale Systems" in *Proceedings of the 15th International Conference on Distributed Systems*, 1995.
 38. D.C. Schmidt and C. Cleeland, "Applying Patterns to Develop Extensible ORB Middleware", *IEEE Communications Magazine*, Special Issue on Design Patterns, April, 1999.
-