

# Integrating Autonomous Enterprise Systems through Dependable CORBA Objects

Carlo MARCHETTI   Antonino VIRGILLITO   Massimo MECELLA   Roberto BALDONI

*Dipartimento di Informatica e Sistemistica*

*Università di Roma “La Sapienza”*

Via Salaria 113, 00198 Roma, Italy

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

**Abstract** – *Integrating autonomous enterprise systems allows the cooperation among entities belonging to distinct systems. As an example, this problem shows up when integrating software services of large departments and organizations of the Public Administration of a country.*

*This paper studies, in the context of the Unitary Network of the Italian Public Administration, the problem of increasing the availability of services exported by an autonomous enterprise system towards others. In particular we show how a fault tolerant CORBA system, namely the Interoperable Replication Logic (IRL), can be used to increase such an availability by building a replicated cooperative gateway that wraps enterprise applications.*

**Index Terms** – High Availability, Object Replication, CORBA, Fault Tolerance, Autonomous and Distributed Systems, Middleware.

## 1. INTRODUCTION

An *autonomous enterprise system* is a collection of many cooperating autonomous systems distributed over a wide, complex computer and communication network, characterized by high heterogeneity in terms of platforms and computing environments that makes difficult the cooperation among remote entities. The *Unitary Network of the Public Administration* (RUPA<sup>1</sup>) is a challenging project promoted by the Italian Government to integrate at the application-level all the autonomous enterprise systems of the Italian Public Administration, that in the past years followed an independent and uncoordinated design and development [2]. In RUPA, heterogeneity of these autonomous systems has been shielded by a *cooperative architecture* [4] that uses application gateways to standardize the access to heterogeneous resources contained in a RUPA *domain*, i.e. RUPA abstraction of an autonomous enterprise systems. Gateways are object oriented wrappers of the domain systems that offer services according to a predefined interface. Being these gateways the access point for all the services exported by a RUPA domain, their fault tolerance and high availability are a key issue in order to guarantee quality of service (QoS) to RUPA clients.

Fault-tolerance and high availability can be obtained by software redundancy: a service can survive to 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 or passive replication (primary-backup approach) [13]. The Common Object Request Broker Architecture (CORBA, [22][25]) is a standard for object oriented distributed applications. It consists of a middleware on

---

<sup>1</sup> The acronym of RUPA comes from the translation in Italian of “Public Administration Unitary Network”, that is “Rete Unitaria della Pubblica Amministrazione” (RUPA).

top of which applications can be designed, implemented and deployed in a very easy way. Moreover CORBA presents high object oriented wrapping capabilities that fits well in the context of heterogeneous environments. 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 [20], OGS [11][12], DOORS [8], Isis+Orbix [16], Electra [16], AQuA [9], IRL[18].
- The Object Management Group (OMG) issued a RFP in 1998 that produced, in early 2000, the Fault Tolerant CORBA specification [10]. 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.

In this paper we analyze how to apply IRL in the context of RUPA cooperative architecture in order to increase the availability of the application gateways by exploiting properties of IRL system such as interoperability and pluggability. Increasing the gateway availability by IRL means increasing the availability of the integration layer and allowing each domain to develop its own cooperative application trusting the availability of the others.

The remainder of this paper is organized as follows: Section 2 illustrates RUPA and its availability requirements, Section 3 describes 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 integration in large-scale autonomous systems. Finally, Section 7 concludes this paper.

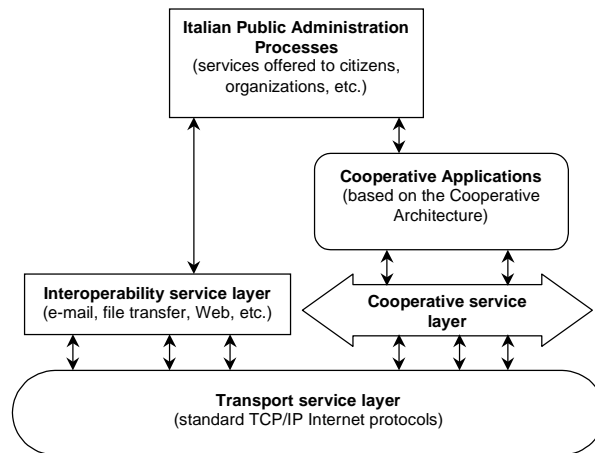
## **2. THE UNITARY NETWORK OF THE ITALIAN PUBLIC ADMINISTRATION (RUPA)**

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] since the beginning of its mandate [1], given by the Italian Government in 1993. RUPA final target 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 make it possible 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: *Transport*, *Interoperability* and *Cooperative Service*. 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*: it includes all the computing resources, networks, applications and data that belong to a *specific* administration, regardless of the technical nature of such information systems.

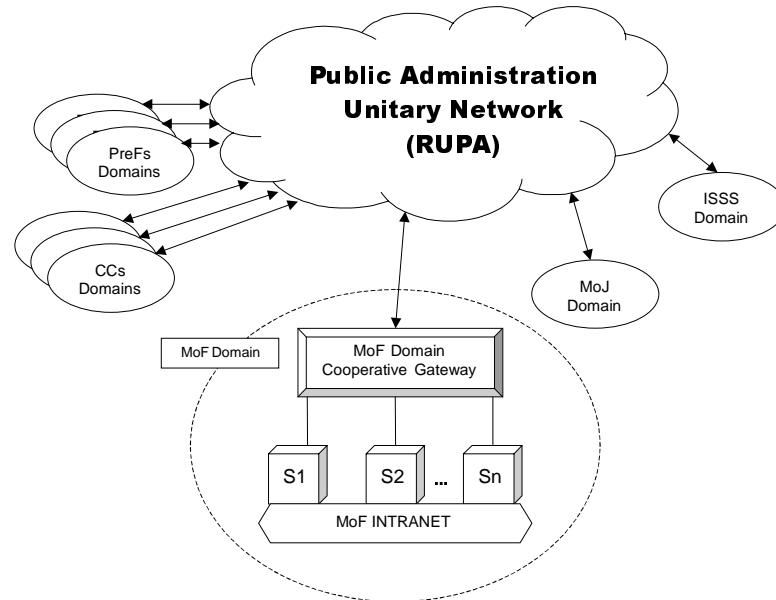


**Figure 1. RUPA Service Layers**

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 – PreFs) it must be developed from scratch (there is no information infrastructure other than simple office automation applications). Thus, in the most general case, *a RUPA domain is an autonomous enterprise 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 application

services described in the export interfaces of the domain<sup>2</sup>. Figure 2 shows the Unitary Network architecture. In this figure, S1...Sn are the heterogeneous MoF information systems, interconnected by a nation-wide area intranet.



**Figure 2: RUPA Architecture**

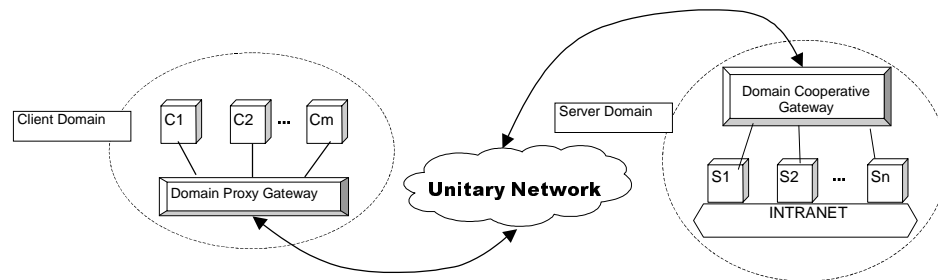
## 2.2. Inter-Domain Cooperation in RUPA

Inter-Domain cooperation in RUPA follows the cooperative architecture model. Interfaces of the services that a domain exports towards RUPA can be specified in accordance with the distributed object computing 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 [19] (e.g. CORBA server objects). Each domain is therefore required to define all the interfaces to be exposed and to implement them in order to make accessible the services they offer.

---

<sup>2</sup> Note that a DC Gateway can be either *centralized*, i.e. constituted by one or a few hosts deployed

End users and applications belonging to a domain interacts with a DC gateway by means of a local *Domain Proxy Gateway* (DP gateway) that has the knowledge of the services available in RUPA and of the protocols and interfaces required to access them (Figure 3).



**Figure 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 the domain must implement both gateways.

DC gateways actually become the middle tier of a layered architecture and carry out the principal 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

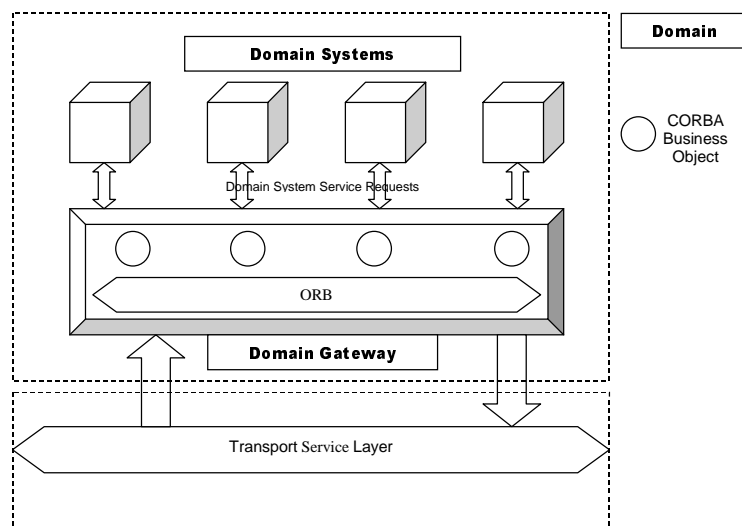
AIPA specifications about the QoS of the Unitary Network ([2],[3],[4]) states that services offered by a particular domain must be highly available, in particular their overall availability must be higher than 99,8%, in order to allow domains to develop cooperative application, each trusting the availability of other domains.

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

---

over a LAN, or *distributed* over a geographical area.

- ◆ **Transport Service Layer Availability.** To maintain connection availability AIPA directives suggest to deploy *redundant ISDN backup lines* at the physical level and to adopt the *dual-homing routing* technique at the network layer.
- ◆ **Domain Services Availability.** In RUPA architecture 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.



**Figure 4: A RUPA DC gateway as a Business Object Container**

- **DC Gateway Availability.** As pointed out in the previous section, DC gateways are business object containers. A business object the object oriented wrapping of a particular domain service provided by a system belonging to a domain. In Figure 4

is represented a domain and its gateway as a business object container<sup>3</sup>. A fault of a particular DC gateway business object is perceived by RUPA clients 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. Thus 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 paid in their design with respect to the fault tolerance and high availability issues.

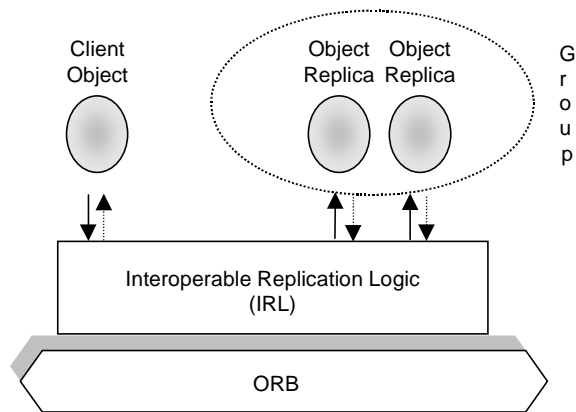
### **3. INTEROPERABLE REPLICATION LOGIC**

Interoperable Replication Logic<sup>4</sup> (IRL) [18] is a CORBA compliant software infrastructure on top of which reliable distributed applications can be built and deployed. IRL has been designed following an “above” the ORB approach (Figure 5), thus remote objects interact through standard IIOP messages without modifying the underlying CORBA platform. However, the price to pay in order to deploy a fault tolerance infrastructure completely “above” the ORB is performance (see Section 5). IRL final target is to develop a FT CORBA compliant infrastructure [10].

---

<sup>3</sup> For simplicity, in Figure we assume a one-to-one mapping between business objects and domain systems, i.e. each business object can perform transactions on one domain system and all the transaction of a domain system are wrapped by exactly one gateway business object.

<sup>4</sup> 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.



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

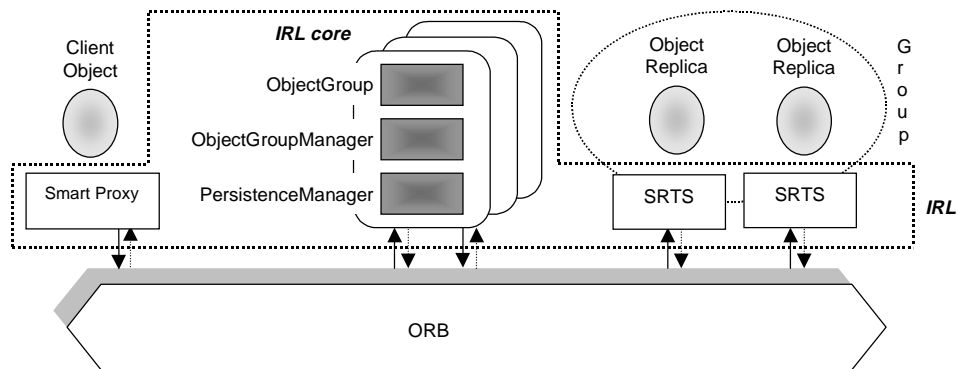
In IRL, the replication logic has been centralized into a single logical component (IRL Core) that does not expose service interfaces (such as multicast communication primitives, failure detectors, etc.). In IRL, a client interacts with a set of server replicas as they were "singleton" objects and server 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 consistency of replicated data. This approach is close to the FT-CORBA specification.

Currently, IRL supports active replication<sup>5</sup> of server objects, running on different CORBA 2.3 compliant ORBs, 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.

In the following section we give a brief architectural overview of IRL architecture. Interested readers can refer to [18] for more details.

---

<sup>5</sup> The portion of replication logic necessary to support passive replication in IRL is currently in the design phase.



**Figure 6. IRL architecture**

### 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 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 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. IRL Core replication design is out of the scope of the paper.

### **3.3. IRL Interoperability and Pluggability**

The primary target of IRL project is to obtain a fault tolerance infrastructure through which it is possible to develop fault tolerant CORBA systems spread over an heterogeneous environment. Moreover, using IRL, dependability can be added also to existing CORBA systems with minimal modifications. Alternative solutions would be either using a different fault tolerance infrastructure for each platform in the system or migrating each system to a common platform, each of which can be very expensive or even impossible (see Section 5).

The CORBA platform independence and wrapping capabilities have been carefully taken into account in IRL design. Thus, as seen in Section 3.1, IRL appears as a collection of CORBA compliant objects logically lying “above” the ORB, i.e. exploiting only IIOP as communication protocol. As a consequence, IRL shows two properties, namely interoperability and pluggability. These properties are described below:

- Interoperability means that IRL can interact with application objects deployed on distinct ORBs. Thus, IRL interoperability allows the deployment of application objects replicas on different platforms running ORBs from different vendors. It is important to note that even each distinct IRL component can run on a different ORB, allowing its deployment in heterogeneous environments such as autonomous enterprise systems.
- With pluggability we mean the possibility of installing IRL over a running ORB without requiring neither to install or configure any other platform dependant service nor to interrupt the service currently offered.

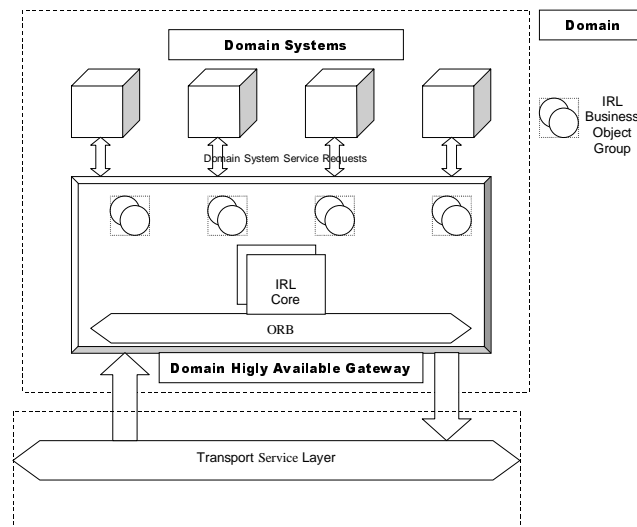
## **4. IRL-BASED HIGHLY-AVAILABLE DC GATEWAYS**

In this section we show how IRL can be applied in RUPA to get high availability of DC gateways.

#### 4.1. IRL in the Autonomous Systems of the Unitary Network

Heterogeneity in RUPA concerns domain systems as well as DC gateways: in order to preserve the autonomy of the administrations, AIPA has neither fixed a particular platform nor an ORB on which deploy DC gateways.

IRL interoperability allows the deployment of the architecture itself as well as of application objects replicas on different ORBs running on different platform. Moreover IRL can be installed on a running ORB without requiring any system stop and restart (pluggability). Deploying IRL on the domain gateways allows to replicate the business object, making them fault tolerant and highly available. Replicating all the business object of a particular DC gateway actually means replicating the gateway and thus increasing the availability of all the domain services. In Figure 7 is depicted an IRL deployment over a DC gateway.



**Figure 7: A Highly-Available DC gateway**

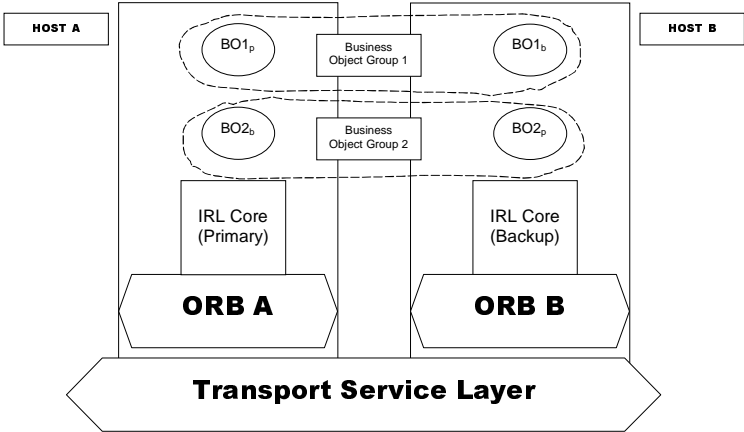
#### 4.2. IRL DC Gateway Deployment

In this section we give an example of a possible configuration of an IRL-based highly-available DC gateway (Figure 8). Let a gateway be composed by two business objects, BO1 and BO2. We need at least two hosts (H1 and H2) to avoid the presence of a single point of failure. The

Business objects are replicated using a passive replication technique. A replicated business object is a business object *group*. Passive replication allows to distribute the two primary replicas on distinct hosts, sharing the computational load<sup>6</sup> between the hosts.

In this example IRL Core objects have been deployed on both the hosts. This is not necessary in general. Actually, business object replicas need only the SRTS (see Section 3.1) to run on their ORBs, while replicas of IRL Core can be hosted on different locations and also on different platforms: the SRTS decouples IRL core deployment from the business object one.

On the client side of the Unitary network, in order to access an highly available DC gateway, the SmartProxy component must be installed on each DP gateway (see Section 3).



**Figure 8: IRL-Based Highly-Available DC gateway Deployment**

---

<sup>6</sup> Active replication would not have allowed such a static load sharing, replicating also the domain service requests coming out from the business objects.

## 5. RELATED WORK

A lot of work has been spent in the last years to add fault tolerance to CORBA applications. As a result, different systems supporting fault tolerance for CORBA application have been proposed in the literature.

A first classification of these systems was proposed by Felber et al. in [12]. In that paper, systems are classified according to the different techniques they use 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 suitability of each system to the heterogeneity and dimension of autonomous enterprise systems, we divide systems into two categories: *Group Toolkit based* and *IIOP based*.

### 5.1. 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 and ordered group communication. Group toolkit based systems relays the task of managing group membership and communication to a group toolkit (i.e. the replication logic, or apart of it, is logically settled “below” the ORB). Systems that falls under this category are described below:

- The Isis+Orbix system is an intrusive extension of the Orbix ORB ([15]) that has been modified in order to add the support for object replication by exploiting a group toolkit (namely, ISIS, [7]).
- In the Eternal system ([20]), 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 the non-intrusion inside the ORB, captures CORBA IIOP messages and “passes” them to a group toolkit (Totem, [21]).

- Also AQuA exploits a group toolkit (Maestro/Ensemble, [14],[26]) 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 replaced by a gateway that passes requests to the group toolkit.

## **5.2. IIOP based Fault Tolerance CORBA Systems**

In the IIOP based systems, the replication logic is logically settled “above” the ORB, and thus the task of managing replica consistency is done all by CORBA compliant objects through IIOP messages. From the description in Section 3, it comes out that 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 provided 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 another COS that, differently from OGS and IRL, implements only passive replication. It requires an application to explicitly subscribe to a COS and define its fault-tolerance properties.

## **5.3. Discussion**

Group toolkit based systems seems not to suit well to autonomous enterprise environments. Group toolkits are proprietary tools, often specialized to a restricted set of platform, and thus they suffer the intrinsic heterogeneity of autonomous enterprises. Moreover, both the geographical distribution of autonomous enterprise systems 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 ([5], [17]) and some theoretical papers ([6], [24]) 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 complex *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 concept is closer to IRL. It doesn’t use group communication protocols, but has a centralized Replica Manager that allows an application object to subscribe to a COS by requesting fault-tolerance services. In IRL, such services are offered to an application object implementing some predefined interfaces. Moreover, DOORS implements only passive replication.

From this discussion it results that IRL and DOORS seems to be more suitable than other systems to be used for integrating autonomous enterprise systems through CORBA dependable objects. However, in small scale system such a *centralized* DC gateway, group toolkit based systems could also be used, probably resulting in performance better than IRL, because of their usage of lower-level network protocols (TCP/IP, UDP etc.)

## 6. CONCLUSIONS

In this paper we have shown how 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 cooperative applications trusting the availability of the business object deployed on the DC gateways of the different domains. IRL design has been also briefly described along with a discussion about the motivations that should promote it in heterogeneous distributed environments. However, in small scale and local system, group toolkit based solutions could accomplish the role of dependable integration platform.

## REFERENCES

- [1] Autorità per l’Informatica nella Pubblica Amministrazione (AIPA): <http://www.aipa.it> .
- [2] AIPA: “La Rete Unitaria della Pubblica Amministrazione”. Roma, Gennaio 1996. [http://www.aipa.it/attivita\[2/reteunitaria\[1/fattib\[1/index.asp](http://www.aipa.it/attivita[2/reteunitaria[1/fattib[1/index.asp) (in italian)
- [3] AIPA-ASSINFORM: “Architettura Applicativa della Rete Unitaria della Pubblica Amministrazione”. Roma, Febbraio 1997. [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)
- [4] AIPA: “Studio di Prefattibilità per l’Individuazione delle Infrastrutture Necessarie a Favorire lo Sviluppo della Cooperazione Applicativa tra Amministrazioni”. Roma, Ottobre 1998. (in italian)
- [5] 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.
- [6] R. Baldoni, R. Friedman, and R. van Renesse, “The Hierarchical Daisy Architecture for Causal Delivery” in Proceedings of 17<sup>th</sup> International Conference on Distributed Computing Systems (ICDCS ’97), Baltimore, MD, 1997.
- [7] K. Birman, R. Van Renesse (eds.), Reliable Distributed Computing with the ISIS toolkit. IEEE CS Press, 1994.

- [8] 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.
- [9] 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.
- [10] Ericsson, Eternal Systems et al., Fault Tolerant CORBA. Joint Revised Submission. OMG TC Document orbos/99-12-19, Object Management Group, Framingham, MA, 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] 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.
- [14] M.G. Hayden, The Ensemble System. Ph.D thesis, Cornell University, Ithaca, NY, 1998.
- [15] IONA Technologies Inc., The Orbix system. <http://www.iona-iportal.com/suite/orbix.htm>.
- [16] S. Landis, S. Maffei, "Building Reliable Distributed Systems with CORBA", Theory and Practice of Object Systems, vol. 3, no. 1, 1997.
- [17] 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.
- [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 2000), Antwerp, Belgium, September 2000.
- [19] 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.
- [20] 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.
- [21] 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.

- [22] 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.
- [23] D. Powell (ed.), Special Section on Group Communication, Communications of the ACM, vol. 39, no. 4, pp. 50-97, April 1996.
- [24] L. Rodrigues, P. Verissimo, "Causal Separators and Topological Timestamping: an Approach to Support Causal Multicast in Large-Scale Systems" in Proceedings of the 15<sup>th</sup> International Conference on Distributed Systems, 1995.
- [25] R. Soley (ed.), Object Management Architecture (OMA) Guide. Second Revision. OMG Framingham, MA, 1992.
- [26] A. Vaysburd, K. Birman, "Building Reliable Adaptive Distributed Objects with the Maestro Tools", in Proceedings of Workshop on Dependable Distributed Object Systems.