A Framework for Inter-ORB Request Level Bridge Construction *

M. Steinder, A. Uszok, K. Zieliński
Institute of Computer Science, University of Mining & Metallurgy
Al. Mickiewicza 30, Cracow, Poland
tel: +48 (12) 17 39 82, fax: +48 (12) 33 89 07, e-mail: {gosia, uszok, kz}@ics.agh.edu.pl

Abstract

The paper addresses a problem of building a bridge between different CORBA compliant systems. It presents a framework of the bridge based on the UNO approach whose architecture is easily extendable to more sophisticated in parallelizing level and functionality units. A problem of mapping objects defined in CORBA model is described and a few suggestions to deal with it are presented. As a case study implementation of the bridge for Orbix and DOME is described.

Keywords

CORBA interoperability, bridge, UNO, framework architecture

1 INTRODUCTION

With the increase of distributed objects' applications a requirement for a common platform grows. This tendency should only strengthen in the near future. Of all the platforms CORBA promoted by OMG seems to gain the greatest interest.

The CORBA Object Model (OMG 93-12-43, 1993) identifies various distribution transparencies which must be supported within each ORB environment. As it has been anticipated, none of the ORB implementations is able to address a large variety of user needs. Existence of many different ORB domains is justified also for performance, security and management reasons.

A diversity of ORBs – now evident – necessitates introduction of means through which they could cooperate; see in (OMG 94-3-1, 1994), (OMG 94-9-32, 1994) and (Uszok, 1994). Interoperability of different ORBs can be viewed as extending standard transparencies to span them. This problem was firstly addressed in CORBA 1.2 specification (OMG 93-12-43, 1993). Later a general architecture for inter-ORB cooperation was put forward in (OMG 94-9-32, 1994), and a notion of a bridge was introduced as a unit residing at a boundary between ORBs transparently transforming requests from a source ORB to a destination ORB.

In this article we aim at presenting a framework for implementing an inter-ORB bridge.

*This work was sponsored by the European Commission under the COPERNICUS project TOCOOS no. CP940247. (http://galaxy.agh.edu.pl/research/cs/TOCOOS/COPERNICUS.html)

By the framework we mean such a conceptual structure of a unit so that it will enable possibly the most ORB-independent implementation. Therefore we divide a bridge into modules according to their ORB dependence. ORB independent parts which access objects via standardized interfaces are uniform, ORB dependent parts which use internal capabilities must be implemented for each system separately. A bridge is created by linking adequate parts.

The framework enables practical evaluation of the UNO approach. It is innately designed for mediated bridges which mediate invocations between domains using a standardized mechanism. We find it however applicable also to implementation of immediate bridges. In our approach Internet Inter-ORB Protocol is used as a mediating mechanism. The framework serves implementing request-level generic bridges whose main functionality – mapping requests between ORB domains – is placed outside ORBs. "Generic" means that bridges enable mapping requests of arbitrary IDL interfaces using dynamic invocation support.

The rest of this paper is structured as follows: in the next section basic assumptions for our design will be presented. Then different half-bridge variants with regard to concurrency models they support will be proposed. In Section 4 functional model of a bridge will be described, in Section 5 a framework for implementing bridges will be presented. Finally as a case study implementations of the half-bridge for Orbix and DOME will be compared.

2 BASIC ASSUMPTIONS

Interoperability may be viewed as extending transparencies to span multiple ORBs. For interoperability between ORBs, ORB services used in the ORBs and the correspondence between them must be identified. The abstract architecture describes ORB interoperability in terms of a translation required when a request traverses domain boundaries. Conceptually, a mapping or *bridging mechanism* resides at the boundary between domains, transforming requests expressed in terms of one domain's model into the model of the destination domain.

In this project **mediated bridging** technique has been taken. In this approach elements of the interaction relevant to the domain are transformed at the boundary of each domain between the internal form of that domain and an agreed, common form. As such a common form, according to the UNO specification, the IIOP protocol has been chosen and used by the "backbone ORB".

Adapting a backbone style architecture is a standard administrative technique for managing networks. It has the consequence of minimizing the number of bridges needed, while making the inter-ORB cooperation match typical network organization.

Construction of a bridge depends on where the bridge components are located: inside or outside ORB. The second option is termed as **request level** bridging. Request level bridges which mediate between distinct execution environments through a common protocol involve components, one in each ORB, known as "half-bridges". Mediated request-level half-bridges can be built by anyone who has access to an ORB, without a need of information about the internal construction of that ORB. It is a main reason why that approach has been adopted to our project.

A general principle of request-level bridging consists in that the original request is passed in the client ORB to an InterORB_Proxy object, which translates its contents to a form that will be understood by the server ORB, invokes the required operation on the apparent server object and passes operation's result back to the client.

Request-level bridges may be: interface-specific or generic. Interface-specific bridges support predetermined IDL interfaces only, and are built using IDL-compiler generated stub and skeleton interfaces. Generic bridges are capable of bridging requests to server objects of arbitrary IDL interfaces using the Interface Repository, Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI).

In this project **generic** request-level bridge is constructed, so new extensions to CORBA specification such as DSI, dynamic typing infrastructure and so on are implemented.

Bridges should support an arbitrary number of InterORB_Proxy objects, which may be created as normal objects using the Basic Object Adapter (BOA) and the DSI. Multiproxy bridge requires internal concurrency of the server process provided by multithreaded environment. It imposes additional complexity on the bridge construction. To separate this factor a single InterORB_Proxy half-bridge has been first designed.

The general architecture of the inter-ORB cooperation is presented in Fig. 1.

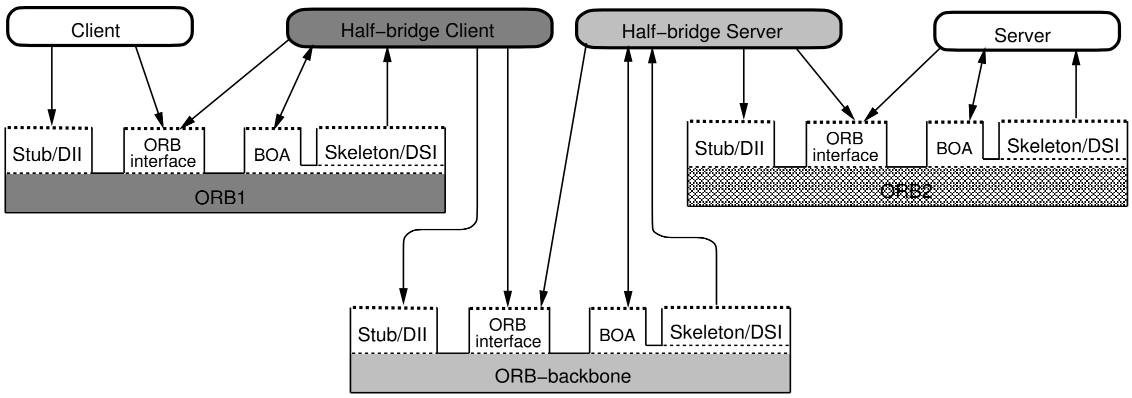

Figure 1 Inter-half-bridge communication scheme

Two ORBs are mediated through the third ORB acting as a backbone ORB. IIOP usage as a mediating protocol implies that a new ORB built around IIOP protocol should be implemented. The minimum functionality of this new environment is determined by the requirements of the generic request-level bridge implementation and is as follows:

- ORB pseudo-object should be supported with extensions concerning initialization, and references comparison,
- Object Adapter functionality for object creation and destruction should be provided,
- DSI and DII interface should be implemented,
- Interface Repository should be available.

Convenient mechanisms for creation and destruction of servers should be also provided. They may be implemented as general ORB modules used for any server creation or as specialized half-bridge factories.

This core ORB functionality should be extended with inter-ORB bridges management layer, that should provide inter-bridge protocol and bootstrapping mechanisms.

3 HALF-BRIDGE VARIANTS

A classical model of object oriented processing approved by CORBA 1.2 specification does not address a problem of internal objects and server parallelism. Its solution is, however, important for multiproxy half-bridges and when object references are used as operation parameters. In this section this issue will be studied in more details.

For efficient implementation of inter-ORB half-bridges it is necessary to exploit a parallel execution of inter-ORB service invocations. Most CORBA compliant software provides multithreading mechanisms as an extension to the basic environment. Its availability depends on an operating system platform. So, the standard solution is in fact single threaded. Therefore in these investigations it was assumed that client and server processes are single threaded.

It has been also assumed that a server is mapped into an operating system process.

In a single threaded environment parallelism may be envisaged only on the level of processes. It leads to a half-bridge-per-remote-server concept where each half-bridge is activated as an unshared server with one active InterORB_Proxy object. This concurrency model will be called *Single-threaded InterORB_Proxy-Half-bridge per Server* and is illustrated in Figure 2.

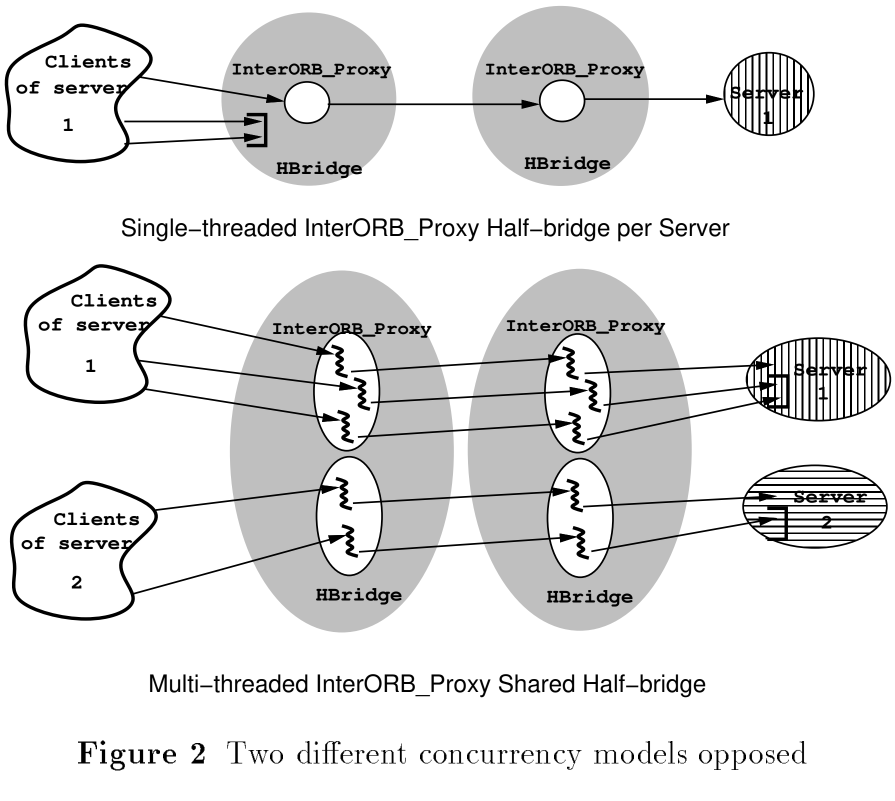

Figure 2 Two different concurrency models opposed

In this model concurrent requests from clients to the same server in the foreign ORB are blocked in a queue in the first half-bridge. This half-bridge represents the server in the client ORB. Queuing requests is a normal activity performed by most of the servers provided in ORB implementations. It is the only way to resolve concurrency problem when a server processes requests sequentially.

In a multithreaded environment three different concurrency models which involve shared server activation policy are anticipated:

- *Multi-threaded InterORB_Proxy-Half-bridge per Server*: for each client's invocation of a service separate thread running through InterORB_Proxy is created. Threads operate in parallel and forward each request to the same sequential server. Forwarded requests are queued in the server instead of the half-bridge as it was in the single threaded model.
- *Single-threaded InterORB_Proxy-Shared Half-bridge*. In this model only one half-bridge server is started in the client and server ORB respectively. For each remote server only one single threaded InterORB_Proxy is created. Requests for the same sequential server in the foreign ORB are queued in the first half-bridge as in the single threaded model. For each server a separate queue that is served by a dedicated InterORB_Proxy must be organized. Inspite of this forwarding a new request when the previous one has been finished is delayed in similar way as in the single threaded model.
- *Multi-threaded InterORB_Proxy-Shared Half-bridge*. In this approach only one half-bridge process is started for all services. For each server dedicated multithreaded InterORB_Proxy is created. This model has similar features to *Multi-threaded InterORB_Proxy Half-bridge per Server* model because requests are queued in the server but it provides light weight parallelism taking advantage of multithreaded implementation.

Without detailed study it is difficult to say what kind of parallelism should be exploited. It will depend on many conditions such as: type of application, availability of multiprocessor machines and so on.

As a starting point the single threaded model has been taken for implementation. It is built of the same components as other models but may be implemented using sequential server supported by most of the commonly available ORBs. It will serve as a basis for future multithreaded implementations which seem to be more efficient.

4 HALF-BRIDGE FUNCTIONAL MODEL

Half-bridge functional model derives from assumptions taken with regard to its location in cooperation environment. Its task is to receive request addressed to the remote server from the local client, translate it into the server's format and transfer it to the server.

To perform this task half-bridge must possess several capabilities: initializing itself, understanding client's request, creating server's request, translating objects defined in CORBA model.

4.1 Half-bridge Initialization

Half-bridge is implemented in a client ORB as a usual server of this environment. It could be activated using original procedures and an object adapter of this environment. After being activated it must also install itself in a server ORB using its original ORB object initialization mechanisms. Then it awaits requests from the client in the client's ORB format.

The proposed architecture of the inter-ORB cooperation assumes existence of the backbone ORB which should be always CORBA2 compliant, since it usually will be connected with not CORBA2 compliant systems. This necessity is fulfilled for instance by the IIOP domain whose implementation is under control.

4.2 Incoming Request handling

A request to the object in CORBA compliant systems is taken over by an object adapter, the same as it was used to activate the object. It uses Dynamic Skeleton Interface (DSI) to pass a request on.

Dynamic Skeleton Interface is a CORBA2 mechanism, therefore currently available ORB systems do not possess its implementation. This necessitates extension of available systems with DSI. Here the general view on how this CORBA part should be designed is presented.

One of the basic DSI objects is ServerRequest defined in (OMG 94-9-32, 1994), which via its standardized interface enables access to the name, parameters and other related data of the requested operation. To build this object client's request must be recognized and matched with the definition of the requested operation containing parameters types. It should be expected that in most ORB systems a request that arrives from a client does not contain information about parameters types inside. In the CORBA standard existence of Interface Repository containing definitions of objects' interfaces has been foreseen. Interface Repository may be contacted to obtain this data.

Functionality of DSI is embraced by *invoke* method of the *DynamicImplementation* object. This function needs an access to operation data including its name, parameters' types and values as well as its result, which are offered through abstract ServerRequest object. Thus the DSI implementor must mostly care how to retrieve CORBA2 typed data from environment specific representation and make it available via this interface. He has also to enable setting values encapsulated inside the object after a call return. When *invoke* completes ServerRequest specified as its argument contains all out and inout parameters and result updated. It is up to the DSI to write them into ORB specific Request object and return to the client.

4.3 Mapping objects defined in CORBA model

CORBA1 standard left some parts of the system undefined because the then state of the art did not allow standardization or some of the elements were intentionally left opaque to allow their specialization for different uses. These deficiencies in the CORBA definition allow vendors of CORBA compliant systems to specify different extensions to the same interfaces to make them usable. In result interface implementation of one ORB cannot be directly ported to the other ORB. In order to construct a half-bridge a mapping from one ORB representation to a representation of the other ORB for all incomplete interfaces must be foreseen. In general to allow two different ORBs to cooperate a mapping from one ORB to another and vice versa must be defined for Objects, TypeCodes, Principals, Contexts and ServiceContext. In the case of a half-bridge built around ORB backbone only the mapping from cooperating environments to this ORB backbone and vice versa is needed. Mechanisms responsible for performing this mapping may take necessary information from bootstrapping or from external protocols.

5 HALF-BRIDGE FRAMEWORK ARCHITECTURE

The aim of this section is to present general approach to constructing half-bridges. An attempt to design such a uniform unit is justified because the presented in the last section half-bridge functionality is immutable. It was recognized that due to the large discrepancies between ORB systems it is impossible to implement a half-bridge able to cooperate with all of them. Instead, a framework is put forward which will serve implementing half-bridges for particular systems. Inside the framework, ORB dependent and ORB independent parts have been distinguished. The former must be implemented for each system separately because they rely on intra-ORB functionalities. The later use only standard CORBA interfaces therefore they may be implemented once for all of the systems.

The architecture of the half-bridge framework is presented in Figure 3.

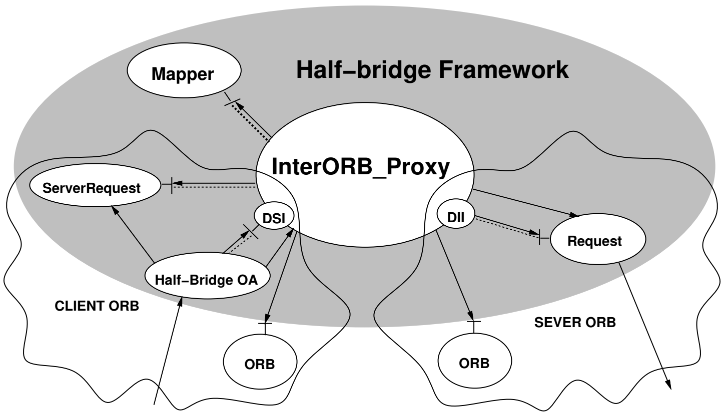

Figure 3 Half-Bridge Framework Architecture

5.1 InterORB_Proxy

An InterORB_Proxy object constitutes a core of the half-bridge framework. It is where the linkage of ORBs with the ORB backbone takes place. The InterORB_Proxy uses standard CORBA interfaces to translate request from the client's ORB to the server's ORB. It possesses dynamic implementation which is a part of client's Dynamic Skeleton Interface. The main InterORB_Proxy's functionality is hidden inside *invoke* method of the *DynamicImplementation* class. It creates a new request performing all necessary mappings and uses server's Dynamic Invocation Interface to forward it.

The InterORB_Proxy is implemented as a template parameterized by names of CORBA modules belonging to adjacent ORBs (Figure 4). These names are in fact half-bridge constants whose values are determined at compilation time. To avoid ambivalence of names for all ORB systems that are to be included in the architecture a new CORBA module is created with a synonymous name. This may be achieved by including an ORB vendor name inside. A new CORBA module will inherit from the old one. The resulting module is extended to be CORBA2 compliant whenever it is reasonable. It contains, for example, a type definition for ServerRequest, extensions to TypeCode interface allowing creating and modifying the object as specified in (OMG 94-11-7, 1994).

It is worth noting that although the InterORB_Proxy possesses some attributes which constitute its state (`PeerRef`, `my_Mapper`) they are never modified while *invoke()* is being performed. The InterORB_Proxy does not remember requests passing through it. Thus it may be considered stateless. Therefore in future multi-threaded version of the half-bridge many threads will be allowed to run through the same InterORB_Proxy simultaneously.

```
template <class CORBA_client, class CORBA_server>
class InterORB_Proxy :public virtual INTERORB_PROXY_BASE_IMP,
                      public virtual DynamicImplementation {
  // It is a stringified object reference of a partner representing a remote server
  char * PeerRef;
  // Reference of the Mapper responsible for translating Objects, Typecodes, Principals,
  // Contexts and ServiceContexts from a CORBA_client to CORBA_server representations
  Mapper * my_Mapper;

  public:

  InterORB_Proxy (CORBA_client::ORB *, CORBA_server::ORB *,
                  REF_TYPE ref, Mapper*) :INTERORB_PROXY_BASE_INIT(ref);
  ~InterORB_Proxy ();
  void invoke ( CORBA_client::ServerRequest *&, CORBA_client::Environment &);
};
```

Figure 4 InterORB_Proxy implementation

Although the main InterORB_Proxy's functionality is performed by *invoke()* method, as the time progress its capabilities will be extended. Firstly the InterORB_Proxy will be equipped with interface enabling monitoring its behavior and management. Then support for firewall capabilities will be developed. Finally it will be integrated with object services: persistence, life cycle and fault tolerance.

5.2 Half-bridge Object Adapter

Creation of the InterORB_Proxy and handling incoming request before InterORB_Proxy's invoke() function is entered is ORB dependent and is performed by a Half-bridge Object Adapter (Half-bridge OA). This part must be created by a modification of a usual object adapter or even built from scratch. When a foreign object reference appears inside a half-bridge this new object adapter has to enable creation of dynamic object – an InterORB_Proxy to encapsulate it if such an encapsulating InterORB_Proxy does not yet exist, giving it an appropriate reference. InterORB_Proxy creation is performed by call to its constructor. The constructor invokes an Object Adapter of the client ORB to register the InterORB_Proxy in it. There are several parameters which must be specified at this time: references to ORB objects of two CORBA systems it connects (the InterORB_Proxy will use some of their functionalities) and reference to an object that will help it to translate certain data eg: object references, contexts etc. (*Mapper*).

When the call to the InterORB_Proxy is recognized the ServerRequest is created and

InterORB_Proxy's *invoke* method is activated regardless of what operation was demanded in the request.

5.3 Mapper

As it was mentioned in the last section apart from forwarding requests mapping objects defined in CORBA is the main functionality of the half-bridge. This task is entirely ORB dependent and is performed by a dedicated object – *Mapper*. The *Mapper* is equipped with *map()* methods – one for each object to be mapped. It reads data from the source object, creates and fills in a target object. There is one *Mapper* in the half-bridge associated with all InterORB_Proxies of this half-bridge.

Generally the *Mapper* is expected to translate CORBA objects between two arbitrary ORBs. In the mediated half-bridge only mapping between ORB and ORB backbone and vice versa is necessary. Since in this project the IIOP protocol is used as an intermediary in the following subsections mapping between any ORB representation and IIOP protocol is discussed.

5.4 Reference Translation

A client which invokes object's operation may place object references as its arguments which denote other objects in the same domain. Such references will not be understandable outside. Therefore they have to be mapped from their proprietary form to an Interoperable Object Reference (IOR) for IIOP. In order to do this we have to fill out the ProfileBody structure (OMG 94-9-32, 1994). The opaque reference form is encapsulated in the *object_key* field. *host* and *port* of this structure are assigned host name and port number of some IIOP domain object which is able to support this reference in the case of calling it. There are two solutions for this problem: eager and lazy mapping. Which of them is used is optional; however, it determines bridge efficiency so it should be tailored to particular applications.

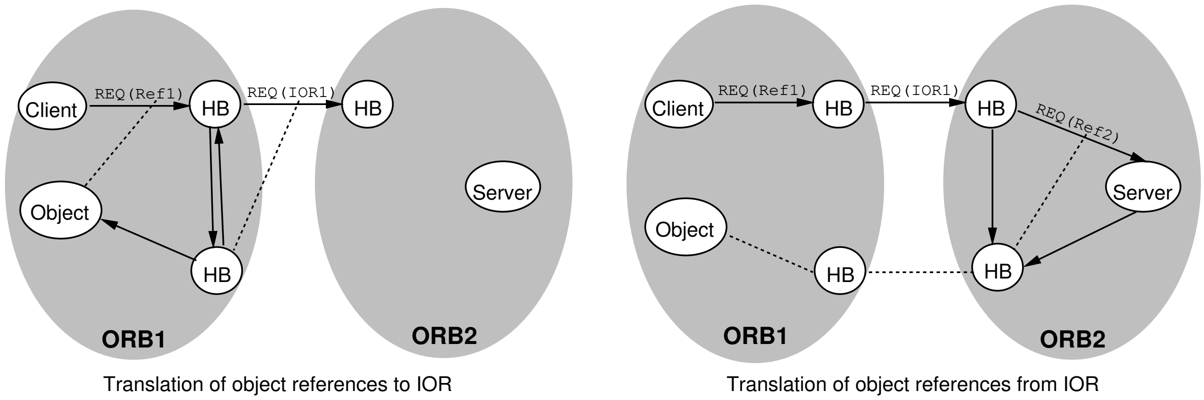

Figure 5 Eager reference translation on IIOP domain borders

Eager reference mapping to IIOP domain
In this approach a new half-bridge is immediately created that will allow contacting object pointed by this reference inside client's ORB (Figure 5). Its IOR including host name and port number is sent to the half-bridge on the server's side. The recipient creates a half-bridge for server environment which will contact their partners in the client's domain. A reference of the InterORB_Proxy inside the newly created half-bridge is sent to the server in the Request message.

Lazy reference mapping to IIOP domain
In this approach a special object – a Bridge Factory is introduced in each cooperating environment (Figure 6). Its host name and port number are used to fill the *ProfileBody* field of the IOR. This object will create a half-bridge responsible for processing all requests to the object whose reference was specified. This action will take place when the LocateRequest message of the UNO approach is received by the Bridge Factory. The LocateReply will contain IOR which points to the InterORB_Proxy inside the created half-bridge.

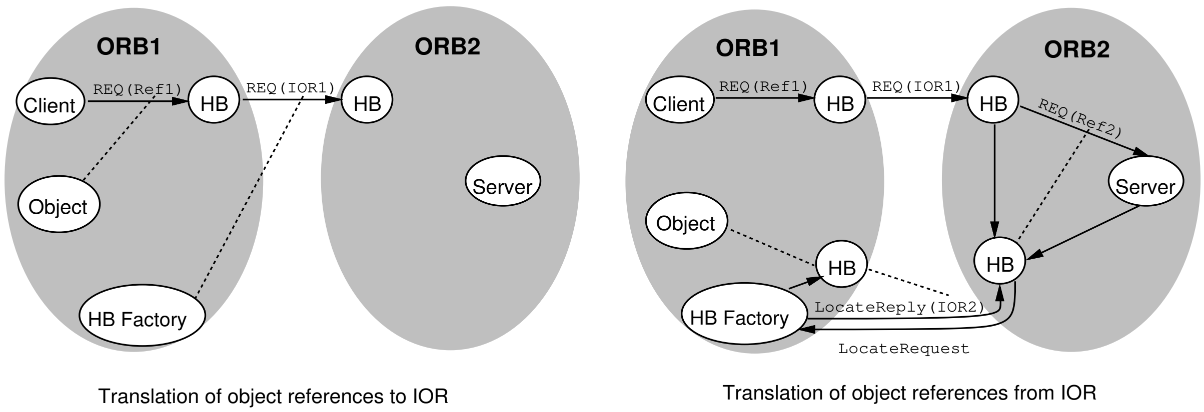

Figure 6 Lazy reference translation on IIOP domain borders

Mapping object references from IIOP domain
A request arriving from IIOP domain to the server ORB may contain IOR references which have to be mapped to the server's ORB proprietary form. The half-bridge must create a new half-bridge with InterORB_Proxy inside which encapsulates the reference. It performs this task using mechanism valid in this ORB. The InterORB_Proxy object possesses a reference specific for this domain which replace IOR in the request. The newly created half-bridge can immediately contact reference it encapsulates to establish a connection or postpone this action until first attempt to use it occurs. When the lazy approach is used the newly created half-bridge firstly sends a LocateRequest message. Reference returned in LocateReply is a final reference to be used during a call. In the case of the eager mapping the original reference is used.

Determining Foreign Object Reference at connection establishment stage
Obtaining foreign references is a quite different problem from that of mapping them. Client existing in certain domain wants to use a server which interface and functionality

it knows but which is implemented in another ORB. In client's domain a half-bridge exists that is able to contact a server within its domain as far as it possesses its reference. The problem consists in finding the server object reference and creating its InterORB_Proxy in client's domain. There are two possibilities to do this:

- bootstrapping
  An InterORB_Proxy for given services is created at half-bridge initialization stage. Information necessary to do that is kept in a persistent database managed by the system administrator.
- trading
  A search for foreign object references and creation of InterORB_Proxies for them is managed by a special trading protocol implemented in the IIOP domain. This may be initiated by a client or transparent for him.
  This additional protocol enables looking up references of the demanded interface (its name is available) inside server's domain with the BridgeFactory as an intermediary. At this stage of the project only a very simple trading mechanism is implemented: each BridgeFactory possesses a list of references of interfaces it exports, a half-bridge uses a LocateRequest message embedding the interface name in its *object_key* field to contact the BridgeFactory, on return it receives a LocateReply message with the object reference inside.

As for now only a single threaded half-bridge is considered, so it possesses either bootstrapping or trading mechanism. A future gateway will possess them both.

5.5 Other CORBA Objects Translation

It has already been recognized that there are other objects that will have to be mapped by the *Mapper*. These are: TypeCodes, Contexts, Principals and ServiceContexts. Although interfaces for the TypeCode and Context were specified in the standard a lot of freedom was left to ORB vendors with regard to their implementation. Corresponding TypeCodes of different ORBs may not be the same in what information they keep inside. CORBA1 standard does not also specify the interface to allow creation of a new TypeCode. Contexts do not give access to all information they hide. As for the Principal no interface for it was specified. The ServiceContext is a CORBA2 notion and is not even mentioned in the CORBA1 standard.

We assume that in all ORBs which use these objects non-standard interfaces exist that will give access to all functionalities not specified in the CORBA1 standard. We may use these mechanisms to retrieve information from them or to write information into them. This however must not be done by the InterORB_Proxy which uses only standard interfaces. Hence translation of this objects is performed by the *Mapper*.

6 CASE STUDY FOR ORBIX AND DOME

The proposed framework has been applied to implementation of half-bridges for two CORBA compliant systems. One of them is Orbix by IONA Technologies Ltd (Orbix, 1995). The system is built with a great conformance to the CORBA standard. The second one is Distributed Object Management Environment –DOME (DOME, 1993). The system lacks many CORBA features and some of implemented ones do not fully conform to the standard. Such different systems have been chosen to stress generality of the proposed framework.

In this section we present some details of half-bridge implementation in these two systems.

Half-bridges in Orbix and DOME are the ordinary servers in these domains. They are launched by means specific for the domain to which they belong. In Orbix the half-bridge may be activated manually as a persistent server. It may also be installed in the Implementation Repository and dynamically activated after a bind() from a client. In DOME there is no Implementation Repository, therefore the half-bridge must be launched by hand.

Half-bridges implemented for Orbix and DOME use the same InterORB_Proxy template and they must care how to register its instance as an ordinary Orbix or DOME object. In Orbix the BOAImpl approach is used to construct an implementation for a given interface. *InterORB_Proxy_base* is an ordinary Orbix interface, therefore Orbix IDL compiler generates *InterORB_Proxy_baseBOAImpl* class for it. The *InterORB_Proxy* - a template inherits from *InterORB_Proxy_baseBOAImpl*. When calling its base class constructor the *InterORB_Proxy* specifies an interface name of the object it represents. This way it ensures delivery of all requests directed to the represented object.

In *DOME* after compiling *InterORB_Proxy_base* interface *InterORB_Proxy_base* and *InterORB_Proxy_base_I* classes exist. *InterORB_Proxy_base_I* inherits from *InterORB-_Proxy_base* and defines dispatching functions for this interface. The InterORB_Proxy template inherits from *InterORB_Proxy_base_I*. In order to initiate itself in the ORB system the InterORB_Proxy must call its base class constructor along with its own. To create InterORB_Proxies *InterORB_ProxyObjectServer* is implemented, that inherits from the DObjectServer class. Its create_object method creates a new InterORB_Proxy for each object constructor it is called by.

To make it possible to use the InterORB_Proxy template the system must possess the ServerRequest interface. This interface has been defined for Orbix and DOME according to the UNO specification. Its implementation requires access to the Interface Repository - a standard CORBA module, which does not exist in DOME. This necessitates creation of this module at least with the minimum functionality. In Orbix Interface Repository exists, is CORBA compliant and may be used to implement DSI without any extensions.

In order to retrieve data from the incoming request in Orbix as well as in DOME DSI uses the streamlike interface of the Request object.

Another ORB specific module used by the InterORB_Proxy is the Dynamic Invocation Interface. In Orbix DII is CORBA compatible and does not require any changes. In DOME only simple implementation does exist, that must be developed and conformed to CORBA.

Half-bridges in Orbix and DOME use also ORB specific mechanisms to deal with an unknown reference. In Orbix when inside a half-bridge a bind to an object for which a mapping has not been recognized yet appears an Object Fault is raised. This activates a *Loader* mechanism. This may be used to browse through some external repositories for this object reference as it was described in the previous section. In DOME when inside the create_object method a name of an unknown object appears it browses through external repositories and causes them to create such an object and return its reference.

7 CONCLUSIONS

In this paper a general framework for inter-ORB half-bridge construction has been presented. The framework was based on the UNO standard and classified as a mediated, request level, generic bridge according to this approach. As an intermediary protocol IIOP was chosen. It constitutes a new backbone domain equipped with all CORBA features which are necessary for implementation of request level bridge.

Four different half-bridge variants with regard to a parallelism level they support have been proposed. As the simplest one the sequential half-bridge has been chosen for implementation. Its architecture has been however designed in such a way so that a progress to more sophisticated variants will be easy.

It has been recognized that due to the incompleteness of the CORBA standard specification and large discrepancies between ORB implementations it is impossible to invent a general bridge connecting arbitrary ORBs. Its implementation depends on them. Therefore it has been divided into ORB dependent and ORB independent parts. A core of the half-bridge – the InterORB_Proxy is ORB independent because it uses only standard interfaces. To the ORB dependent parts belong: ServerRequest implementation as a CORBA2 extension to a client ORB, a Half-bridge OA - a modified or new object adapter for InterORB_Proxy creation and request handling, and a Mapper responsible for translation of CORBA objects into IIOP and vice versa.

As it has been noticed the greatest problem in implementing inter-ORB half-bridges is mapping. This is why this task was imposed on a separate object - the Mapper. Mapping is entirely ORB dependent. Only general assumptions may be given as for global solutions. In this article we have presented how object references should be mapped and obtained.

In the last Section application of the framework to two distinct CORBA standard implementations: Orbix and DOME was described. Most of this experience will be used in the future when other systems will be added to the architecture.

8 REFERENCES

CORBA 1.2 Revision Draft (1993) OMG Report 93-12-43, Object Management Group (OMG) Inc.

W. Harrison, *The Importance of Using Object References as Identifiers of Objects – Comparison of CORBA Object References* (1994) IBM Watson, TR

ORB Interoperability. Joint SunSoft / Iona Submission to the ORB 2.0 Task Force Initialization & Interoperability Request for Proposals (1994) OMG Inc., TC Document 94-3-1

Universal Networked Objects (1994) OMG Inc., TC Document 94-9-32

Interface Repository (1994) OMG Inc., TC Document 94-11-7

ORB Initialization Specification (1995) OMG Inc., TC Document 94-9-46

A. Uszok, G. Czajkowski, K. Zieliński, *Interoperability Gateway Construction for Object Oriented Distributed Systems*, (1994) Proceedings of 6th Nordic Workshop on Programming Environment Research,

Orbix Programmers Guide (1995) IONA Technologies Ltd.,

DOME User Guide (1993) Object-Oriented Technologies Ltd.,

9 BIOGRAPHIES

Małgorzata Steinder graduated from the Computer Science Department, University of Mining and Metallurgy in 1994. Her MSc thesis concerned mathematical models describing performance in the ATM network. Currently, she works as a research and teaching assistant in the Computer Science Department, UMM Cracow. She is interested in interoperability and trading in distributed processing. She is working on her PhD thesis.

Andrzej Uszok graduated from the Computer Science Department, University of Mining and Metallurgy in 1993. His MSc thesis concerned load-balancing in the ANSA system. Currently, he works as a research and teaching assistant in the Computer Science Department, UMM Cracow. He is an author of 11 articles in the area of distributed systems. He has just finished his PhD thesis: *Transparent Interoperability in Distributed Processing.*

Krzysztof Zieliński is a professor of computer science at the Computer Science Department, University of Mining and Metallurgy. He spent two years in the Cambridge Olivetti Research Lab in the years 1988-1990, where he worked on the first prototype of the ATM network. Now, he is the main designer of the Cracow ATM MAN. He is also the Technical Manager of the Copernicus **TOCOOS** project. He is an author of about 100 publications.