A CORBA-Based Surrogate Model on IP Networks

Genaína N. Rodrigues[1], Carlos A. G. Ferraz[2], Sérgio V. Cavalcante[2]

[1] Dept. of Computer Science, University College London - London, UK

[2] Centro de Informática, Universidade Federal de Pernambuco (UFPE) - Caixa Postal 7851, 50732-970, Recife-PE, Brazil

g.rodrigues@cs.ucl.ac.uk, {cagf, svc}@cin.ufpe.br

Abstract. *In order to provide ubiquity and therefore a wide communication approach, many researchers have exploited solutions for fitting distributed system profiles into small devices with stringent memory requirements and low CPU speed. In this regard, research in the field of middleware for embedded systems has proliferated. In this paper we present an architecture that allows devices with limited memory resources participate in a distributed system network by means of a middleware framework. Through our model, devices with quite stringent memory requirements may become part of a network of CORBA technology-enabled services. In particular, we illustrate the working model by supporting IP-based network devices*

1. Introduction

For many years, embedded systems were thought to be elementary devices with limited functionalities without regarding a wider communication approach. Nevertheless, this scenario has been changing with the aid of telecommunications research and the increasing interest in a simple and efficient way of connecting people to a wide range of services.

Essentially, distributed systems provide this infrastructure which programming complexities are tamed by object-oriented middleware at a certain level. Those middleware not only realize a modular 3-tier client-server model but also leverage concerns to be addressed in the design of non-functional requirements. Among others, some types of object-oriented middleware have been extensively used, like CORBA (Common Object Request Broker Architecture) [7] and Jini™ Network Technology [13].

Many researchers are trying to find out the best way to fit distributed system profiles into devices with stringent memory requirement. Three major middleware trends can be noticed: shaping middleware profiles for specific devices, reducing memory footprint of heavy-weight middleware and building components that act on behalf of the device inside the distributed framework.

Essentially, the first tendency is supported by the MinimumCORBA Specification [8] from the Object Management Group (OMG) and the J2ME project [12] from Sun Microsystems. The second tendency can be represented by device-specialized CORBA profiles such as PalmORB [10] and the like. The third tendency suits properly the Jini Surrogate Project [16].

By and large, CORBA is a well-established standard and an interoperable environment with support for various programming languages. On the other hand, the ability to support mobile Java code and the wide adoption of Java make Jini an attractive possibility.

Considering that each application developer requires a different subset of the standard middleware features and services, like the one Jini Surrogate supplies, joined to the support for multiple languages and platforms stemmed from CORBA, this work proposes a CORBA Surrogate Model called UORB (Ubiquitous Object Request Broker).

Furthermore, this work aims at providing resources for the participation of devices into a CORBA environment without requiring a CORBA profile inside the device. In other words, the CORBA heavy-weight processes would occur in a host other than the device.

This paper is structured in the following sections: in section 2, we outline the major concepts so as to provide the background. In section 3 we present our designed and implemented solution to allocate embedded systems with restricted memory concerns into CORBA. In section 4, we present the implemented model and exploit the qualitative concerns of the working model compared to the Jini Surrogate architecture, proposed by Sun Microsystems, and the MinimumCORBA proposed by OMG. In section 5, we finally conclude and raise some future extensions to our Surrogate CORBA model.

2. Background

In order to better support a wider range of services where distributed-architecture frameworks could suit not only personal computers but also devices with stringent memory requirements, Distributed Embedded Systems (DES) [4] have emerged as a feasible solution, and a middleware framework naturally stems from them.

Considering that the major concepts of our project are tightly related to those defined in the Jini Surrogate Architecture, we outline this architecture in the following subsection.

2.1 The Jini Surrogate Architecture

In order to join small devices in a Jini network, there are some requirements that must be satisfied like the ability of those devices to participate in the Jini discovery protocol. Also, downloading and executing Java classes would be required. Nevertheless, there is a category of devices that cannot fulfill these requirements. Therefore they cannot participate in the Jini Network.

By defining a way these devices can participate in that network, the Jini Technology Surrogate Architecture [16] addresses those problems by stating the following requirements:

1. Device Type Independence – The surrogate architecture must be able to support a wide range of devices with different capabilities.
2. Network Type Independence – This independence means that the surrogate architecture must support different protocols even when they are simultaneous on the same physical network.

3. Preserve Plug-and-Play – This architecture keeps the concepts that stem from Jini concerning discovery, code downloading, and leasing of distributed resources.

Before presenting an overview of Jini Surrogate architecture, some important concepts regarding that architecture are shown below [16]:

- The term *device* refers to a hardware or software component that is not capable of directly participating in a Jini federation;
- A *surrogate* is an object that represents a device within the Jini network. The implementation of the object may be a collection of Java software components and other resources;
- A *host-capable machine* is a system that allows the downloading of code and the running of a surrogate, as well as comprises a Jini federation. Additionally, the host-capable machine is accessible to the entity offering the surrogate;
- The *surrogate host* is a framework that resides on the host-capable machine and provides a Java application environment for executing the components of the surrogate architecture. In addition to providing computational resources, execution environment and life-cycle management, the surrogate host may also provide other *host resources* to assist the components in the architecture;
- An *interconnect* is the logical and physical connection between the surrogate host and a device. There may be more than one interconnect defined for a physical connection. It is also possible for the interconnect to be on the same physical connection that forms the Jini federation;
- The s*urrogate protocol* includes the interconnect-specific mechanism for discovery, retrieval of the surrogate, and device reaching; and
- An *interconnect adapter* is an interconnect-specific component or a set of components that works with the surrogate host that implements the surrogate protocol.

Overview

In order to realize the surrogate architecture, there must exist a host-capable machine connected to the device interconnect and the Jini network. Furthermore, the host-capable machine must be able to execute code written in Java on behalf of the device and supply the resources that such code may need.

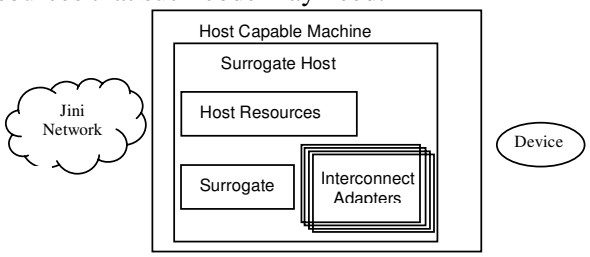

Fig. 1. Jini Surrogate Architecture Components

Initially, the surrogate host dwells in a host-capable machine. In fact, the surrogate host can be seen as a Jini gateway. Moreover, there is an interconnect adapter monitoring the device interconnect. It is possible that a single surrogate host can contain more than one interconnect adapter. The Jini Surrogate Architecture components are briefly illustrated in Fig.1.

Device Discovery and Surrogate Loading

The interconnect adapter is in charge of implementing the surrogate protocol for a specific interconnect.

The first task of that protocol is to use one that enables the device and the surrogate host to find each other. This is done by the Discovery protocol that performs similarly to the one that compounds Jini. A discovery protocol is specific to the Interconnect and varies according to capabilities of the Interconnect Adapter and the device.

Afterwards, the surrogate of the device must be retrieved. This can be performed by either operations: push – the device uploads the surrogate to the interconnect adapter, or pull – the interconnect adapter must extract or download the surrogate from the device. The bytecode and other relevant data are stored in a compressed jar file.

An instance of the surrogate is then created by the surrogate host and executed in the context of the surrogate.

Surrogate Execution

When execution is being performed, the surrogate can accomplish any task necessary for the device, including the access to the Jini network. Additionally, the surrogate may also communicate back to the device using any appropriate protocol.

Once a surrogate is loaded and activated, both the surrogate and the interconnect adapter will monitor the reaching of the device. It means that a convenient communication path exists between the device and its respective surrogate. Despite the Jini Surrogate facilities, the realization of a CORBA Surrogate Model seems to be rather appealing, which is reasoned in the section 4.

3. Related Work

On the whole, middleware is a commonly proposed solution to denote a set of generic services above the operating system. In the context of this work, the term middleware is mostly related to middleware designed to ease embedded systems development by providing a component-based framework. Besides the specifications of MinimumCORBA [8] and Jini [13], examples of those kind of middleware include Zen [3] and LegORB [11]. These contributions have been successful to address flexibility in adopting policies and reflection mechanisms. However, as our focus is to make viable the use of distributed network resources to legacy limited-memory devices, we believe that CORBA can provide the required infrastructure. Additionally, the standardization of CORBA makes it suitable for interoperability issues.

Based on the fact that devices require only a subset of the whole distributed systems infrastructure to use the middleware facilities, some contributions have targeted to design middleware profiles for specific devices. Particularly, it has been noticed that these contributions [10] have been primarily made upon CORBA kernel implementation (i.e. ORB). However, as their designs are specific for flavors of devices, their extensibility might degrade over the years. Additionally, as far as non-functional aspects (e.g. concurrency, persistence, synchronization) are regarded, it would probably be a tough task to integrate a network of different kinds of devices with specialized ORB profiles.

4. The CORBA-Based Surrogate Model

On the whole, this project aims at providing an architecture to allow devices with limited memory resources participate in a distributed system network by means of a middleware framework. Through our model, devices that cannot easily take advantage of a CORBA environment, for instance, may become part of a network of CORBA technology-enabled services, or devices, or both, and other distributed networks through interoperability facilities.

4.1 The Benefits of a CORBA Surrogate

There are some reasons that led to the decision of which architecture could be first used in order to implement distributed network support for limited devices. For one thing, the CORBA architecture is a standard, what makes it easier to interoperate with a broader range of applications. Unlike Jini, it is a cross-platform communication architecture, which enables definition, transportation, implementation, and invocation of objects in many programming languages such as C, C++, Smalltalk and Java.

Additionally, CORBA can offer support to a wide range of languages. Moreover, despite the fact that CORBA was not originally tailored to provide QoS features, recent additions to the CORBA specification, such as Real-time CORBA and CORBA Messaging, address many end-to-end quality-of-service aspects. These specifications standardize interfaces and policies for defining and controlling various types of applications QoS aspects [5].

For the other thing, the higher level of abstraction proposed by the Sun's Surrogate model can bring such a flexibility that devices could participate within a CORBA network without implementing a CORBA interface. Although this flexibility centralizes CORBA operations in the Surrogate Host, it saves the device from greater resource allocation problems.

Compared with the MinimumCORBA approach, the design of individual devices implementation objects in the Surrogate model may be more complex. However, the number of devices that can be served by the Surrogate network-available proxy is not limited by the physical constraints of the device. Furthermore, there is no requirement that the device and its proxy be in the same physical host.

In order to analyze the elements comprised by the CORBA Surrogate Model, the following features arise:

- Devices as non-CORBA objects;
- Pluggable Devices;
- Pluggable Network Adapters;
- Dynamism and other Original CORBA Services;
- Support for a Surrogate Host Federation.

4.1.1 Devices as non-CORBA objects

Unlike the MinimumCORBA approach, the CORBA Surrogate Model does not implement any CORBA interface within the device, and, therefore, it makes no assumptions about any CORBA Service. In fact, devices make use of the CORBA

infrastructure by means of Surrogate objects that act on behalf of the device in the CORBA network.

Considering that a Surrogate object stands for a device in the distributed network, this object in the CORBA Surrogate model has a unique object reference. Furthermore, the participating Surrogate objects may be implemented in a variety of ways using different programming languages.

Whether a device intends to communicate with other CORBA objects, the Surrogate object acting on behalf of that device must know the CORBA object contract and its semantics through the Interface Definition Language (IDL), and the server component reference through the object reference (IOR) as shown in Fig. 2.

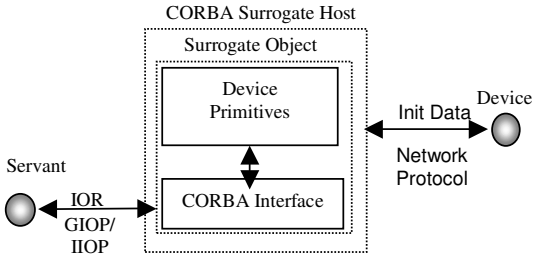

Fig. 2. General View of the Interaction Between a Device and the CORBA Surrogate Host

4.1.2 Pluggable Devices

To support the plug-and-work model [9], devices can seamlessly join and leave the CORBA network without incurring troublesome communication errors. This feature is part of the discovery mechanism between the device and the Surrogate Host, described in the forward **The IP Interconnection Protocol** section. Whenever a Surrogate Host shows availability to receive incoming device requests via multicast messages, the Surrogate object standing for the device is registered in the Surrogate Host, enabling the Surrogate to be loaded and activated. Devices that own the surrogate object reference can then obtain a server reference from the CORBA Naming Server or the ORB itself and instantiate the server proxy.

The device, by means of the Surrogate object, knows the server id, but needs no explicit knowledge of the server's physical address or port. Thus, it follows the highly configurable framework of distributed object systems. A server may live on the same Surrogate Host machine or it may reside anywhere on the network. Additionally, this feature ensures that changes can be made to the Surrogate object and that the correct drivers are used the next time the device intends to participate within the CORBA network.

4.1.3 Pluggable Network Adapters

This feature makes the CORBA Surrogate a flexible model such that network-adapted devices attached to a specific protocol can be supported, since its protocol interconnect adapter has been previously implemented. This interconnection may be related to logical or physical connection (e.g. IP network and USB port) between the CORBA Surrogate Host and the device.

As far as the protocols to support the communication between the device and the CORBA Surrogate Host are regarded, the latter can be connected to as many interconnect adapters as GIOP could support. For example, TCP/IP, shared memory, UNIX-domain sockets, and SSL could be the network environment used by the device to communicate.

4.1.4 Dynamism and other Original CORBA Services

According to [3], when specific ORB features are omitted, as described in the MinimumCORBA Specification, the result is to significantly reduce the suitability of the middleware for general use. For instance, if an application requires a small footprint, it might use servant activators, servant locators, or even a default servant to help reduce application footprint. However, all these features, specially the CORBA dynamic features, have been removed from MinimumCORBA.

In the CORBA Surrogate model, all CORBA Services are kept in order to better make use of the CORBA infrastructure. In fact, the CORBA Naming Service and the Implementation Repository (IMR) are primarily required. Other CORBA Services, such as Trading, Event, Notification [6], and the dynamic features (i.e. DSI and DII) as well as the Interface Repository, may also be supported.

4.1.5 Surrogate Hosts Federation

The Surrogate Model can be included in the range of smart networks with respect to its infrastructure ability to easily discover and utilize the services of other network participants, and ability to adapt to unpredictable changes in network membership, throughput, and latency.

Whether a large-scale support is required, a Surrogate Host Federation can be conceived in order to meet performance, scalability, availability and fault-tolerance requirements. Regarding availability and fault-tolerance, the system should keep running even in the presence of partial failures, searching for alternative running instances in place of failed objects and dynamically binding them to the federation. Regarding performance, the federation should be able to achieve an acceptable overall efficiency by means of conscious balanced compromises.

As far as scalability is concerned, the federation should be able to grow, adding new surrogate and service instances, name servers, and hosts to support high volume of access and registration. In this regard, the objects in the ORB could run on multiple servers to provide load balancing for incoming device requests. Then the ORB could dispatch the request to the first available object and add more objects as the demand increases.

4.2 Resources Required

In this section, the requirements of the device and the CORBA host to participate in the CORBA Surrogate Model are outlined.

4.2.1 The Device Requirements

The surrogate architecture lowers the requirements that a device must meet to participate in a distributed network. The surrogate architecture provides a place for objects implementing the device interface to exist other than on the device itself.

Accomplishing the same Jini Surrogate philosophy, one of the most notable strengths of the Surrogate Architecture is that it does not depend on any particular capability of a device managed by a surrogate. Devices that have severely limited processing power, limited memory, or primitive communication capabilities can still participate in the surrogate architecture. In fact, all of the intelligence can be built into the network proxy (i.e. Surrogate object), perhaps uploaded into the proxy by the service device, following the same approach adopted in the Jini Device Architecture Specification [14].

Considering that the Surrogate classes do not run in the device itself, they may not reside in the device and therefore there is no limitation regarding their size. The API the device requires to communicate with the CORBA host isn't a limitation either, except for the need to have an interconnect adapter to provide the communication between both parts.

To sum up, the device should only implement the interface to communicate with the Interconnect Adapter. In the case of the Jini Surrogate Model, the device could comprise the classes that implement the surrogate object or it could contain information to locate this surrogate in the network. On the other hand, the CORBA Surrogate Model is based on the second solution.

4.2.2 The CORBA Host Requirements

The requirements to have all CORBA Surrogate Architecture modules running are the same required to any other CORBA servant. In other words, the host machine targeted to run the Surrogate Host framework should support a 32-bit operating system as well as memory resources to run a programming language such as Java or C++.

So as to locate, activate and execute Surrogate objects, the CORBA Naming Service and the Implementation Repository (IMR) are required. Both requirements should be initialized before any module constituting the CORBA Surrogate Architecture.

Within the CORBA Surrogate Model, the IMR is initially required to store and retrieve the surrogate object implementations. Additionally, this repository provides other features such as the activation and deactivation of objects and their servers.

Within the CORBA Surrogate Model, the Surrogate Host and the Adapters are required to be registered in the Naming Service. In the next section, we present the major designing details of UORB and the modules it constitutes as well as their interactions.

4.3 The CORBA Surrogate Modules

As for design details, the CORBA Surrogate Model embraces two major modules: the Host and the Adapter. The former supports the loading, activation and execution of the Surrogate object. The latter supports the communication between the Surrogate Host and the device.

4.3.1 Overall Architecture

Through the Surrogate architecture, devices that are not directly connected to a distributed network, or are otherwise unable to have direct access to the distributed

technology infrastructure, can supply surrogates that can access the distributed network and have access to its infrastructure. These surrogates act on behalf of the device that cannot embrace distributed technology support. The CORBA surrogate implementation aims at permitting devices make use of the CORBA infrastructure.

In order to fulfill the requirements of a Surrogate Model, the project has two greater parts: one concerning the Surrogate Host and other concerning the communication between the device and the Surrogate Host.

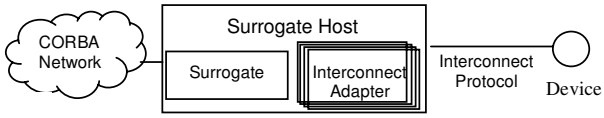

Fig. 3. UORB Modules

4.3.2 Surrogate Host

The Surrogate Host is a framework that resides on the machine that can afford the computational resources required to load, activate and execute CORBA objects. The Surrogate Host also provides an application environment for executing the components of the surrogate architecture. In addition to providing computational resources, an execution environment and life-cycle management, the Surrogate Host may also provide other host resources to assist the components in the Surrogate architecture.

4.3.3 Interconnect Adapter

The Interconnect Adapter is the means the device utilizes to communicate with the Surrogate Host. The Interconnect Adapter (or merely Adapter) can be viewed as a gateway to let devices reach, establish communication and make use of the Surrogate Host framework, and thus, of the CORBA network. The communication between the Interconnect Adapter and the device is supported by the *interconnect protocol*.

The term interconnect protocol defines the mechanisms for discovery, surrogate retrieval, and liveness of the surrogate object representing the device. According to the Jini Surrogate Architecture Specification, for each interconnect there must be an interconnect specification that fully describes discovery, surrogate retrieval, and liveness.

The interconnect protocol requirements to support the communication between a device and the Surrogate Host are described in the following.

5. The CORBA-Based Surrogate On Top of an IP Network

In this section, the implementation of the UORB framework is presented. The implementation of the UORB models was primarily based on the Jini Surrogate Specification along with the IP Interconnect Specification by Sun Microsystems group.

5.1 The IP Interconnection Protocol

IP can be implemented for various physical media such as Ethernet or wireless networks. Any physical layer that supports IP networks can employ the infrastructure of the IP interconnect protocol and therefore the auxiliary protocols it requires to realize the communication between devices and the CORBA Surrogate Model components.

To some extent, the mechanisms supported by this interconnect protocol are discovery, retrieval and liveness (see **The Jini Surrogate Architecture** section). Moreover, each of these mechanisms embraces one or a set of protocols defined to better accomplish its target.

In this section, these mechanisms are presented as well as their respective protocols, primarily based on [15], in order to exchange information between an IP-based device and the CORBA Surrogate modules.

5.1.1 The Discovery Mechanism

By means of the discovery mechanism, the surrogate host and the device are able to find each other on the interconnect, without requiring specific location information. To carry this out, a surrogate host and a device must support two multicast protocols:

1. Multicast host announcement protocol – the surrogate host uses this protocol to announce its availability to receive device requests over the interconnect; and
2. Multicast host request protocol – the device uses this protocol to discover a surrogate host somewhere in the domain of the LAN the device is connected to.

The Multicast Host Announcement Protocol

Via the multicast host announcement protocol, the Surrogate Host performs the action of announcing its presence on the interconnect and indicates its availability to host surrogates. A device listens for the multicast host announcement and, as soon as it receives the announcement from the surrogate host, the device sends a registration request to the surrogate host.

The Multicast Host Request Protocol

Another protocol that is part of the discovery mechanism of the IP Interconnect Adapter is the multicast host request protocol. This protocol is used by a device to locate a Surrogate Host, while the multicast host announcement protocol is used by the Surrogate Host to announce its availability.

The multicast host request protocol comprises two major phases. The first is the sending of a multicast host request from the device to the surrogate. The second is the response the surrogate host, listening on the multicast port, sends the device via a unicast host reply. Then the device uses the address in the unicast host response to generate a registration request for that host to serve as host for its surrogate. If the host accepts the request, the surrogate can be finally retrieved.

5.1.2 The Surrogate Retrieval Mechanism

The retrieval of the surrogate is realized by means of the registration request protocol. This protocol defines rules and message formats the device uses in order to request the load of a surrogate through the Surrogate Host.

By sending a registration message, which contains the location of the surrogate, the device makes the request of the surrogate retrieval to the Surrogate Host. In addition, the registration message may contain data required to the surrogate initialization.

The Registration Request Protocol

Through the discovery protocol or through configuration information, the device obtains information to request the retrieval of the surrogate, i.e. the IP address and the port of the Surrogate Host. The request may be accomplished by sending a unicast UDP message via a datagram packet or sending a TCP message via TCP socket connection. Both messages have the same format.

5.1.3 The Surrogate Liveness Mechanism

This mechanism is in charge of determining whether the device is still reachable on the interconnect. It is comprised of two components: the device presence and the surrogate residence.

The first component tests continually if the device is still present on the interconnect. If the device is no longer present, became non-responsive, or suffered any other failure, the surrogate standing for that device is deactivated. The Surrogate Host sends messages to the device via Interconnect Adapter from time-to-time to test if the device is still alive.

The second component is the one which describes the responsibility of the surrogate to inform to the device if that surrogate is still hosted by the surrogate host. The implementation of this feature is realized by a time-based solution through the keepAlive method of the keepAliveHandler interface below. The KeepAliveHandler interface is the means by which the livenes of the surrogate is monitored.

```
interface KeepAliveHandler{
    void keepAlive(long period);
};
```

5.2 The CORBA-Based Surrogate Model Implementation

In order to realize the communication between a non-CORBA peer with a CORBA framework, a considerable effort was required. However, with the aid of a properly structured architecture, an extensible and changeable system was carried out. In this section, the design and implementation concern of the UORB model are explored in order to have a clear view of the development of this project.

First, the architecture design of the packages, i.e. the logical units regarding the implementation task, and their interactions are illustrated. Afterwards, a detailed design of the main packages comprising the classes that implement the UORB model is outlined.

5.2.1 Architecture Design

With the purpose to have a well-designed architecture, the logical units of UORB can be seen as the interaction between the following packages:

- UORB – Contains the classes implementing the infrastructure to accommodate the surrogate(s) of their respective devices. This package cooperates with the IP Adapter package, which implements the primitives for interconnection between a device and the Surrogate Host in an IP network;

- Simulated Device – Implements a simulated device that is capable of communicating with IP networks. It should be noticed that, although the Fig. 4 shows the straight connection between the Simulated Device package and the IPAdapter package, they do not instantiate each other. The details of this communication is accomplished by the means of IP interconnect protocol presented in **The IP Interconnect Protocol** section;
- IPSurrogate – Implements the functions that allow the surrogate object act on behalf of the simulated device within the CORBA Surrogate Host. This package interacts not only with the Simulated Device package, but also with the UORB package because the surrogate must be previously instantiated (or marshaled in the CORBA point of view) and activated through the Surrogate host;
- IPAdapter – Makes feasible the support of UORB to IP networks. With the facilities implemented in this package, a device that is part of an IP network can communicate with the CORBA Surrogate Host framework;
- ThreadPool – Implements a thread-pool concurrency model in order to provide a reliable and efficient communication in the distributed environment of UORB. Basically, it deals with issues such as the amount of threads the program requires, the management of the threads and so on. The implementation of this package is not vital to accomplish UORB, but desired; and
- Utility – Contains classes to aid in the development of certain classes where general functionalities could be used throughout the packages. Classes to parse command line arguments or classes to provide a unique reference (the Singleton design pattern [2]) of the ORB as well as the CORBA Naming Service are instances of these classes.

The design of these packages is shown in Fig. 4.

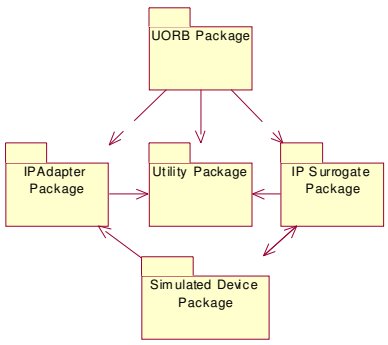

Fig. 4. Application Packages of UORB and their dependencies.

5.3 Trade-offs between the Surrogate Models and the MinimumCORBA

In order to find out what solution best fits one's purpose, an evaluation of UORB and similar architectures is necessary. In the range of general solutions aiming at providing middleware for embedded systems, UORB, the Sun Surrogate reference implementation (Madison) and the MinimumCORBA can be compared at a certain level. In this case, the MinimumCORBA implementation chosen was the Orbix/E 2000 from IONA™, which is the same vendor of the CORBA implementation that UORB was developed, i.e. ORBacus version 4.1.0.

This section proposes a comparison of these approaches:

- the size of the generated classes to embed in the device;
- the memory resource the device should provide to run those classes; and
- qualitative concerns.

There are two situations concerning the classes to embed in the device: as a client and as a server. Each of these situations must be considered individually so as to properly treat the situation being analyzed. In order to gather the minimum resources required by the MinimumCORBA, the application implemented in the Orbix/E was a simple "Hello World" sample.

Finally, it should be mentioned that all these classes were implemented in Java programming language, i.e. JDK version 1.3. An advantage stemmed from Java language that naturally arises from this is the broader portability achieved.

5.3.1 The Size of the Classes

In order to measure the size of client and server examples classes were considered.

Table 1. The Bytecode Size of the Classes Generated to Embed in the Device

	Client	Server
UORB	45 KB	45 KB
Madison	52 KB	52 KB
Orbix/E	100 KB	150 KB

From Table 1, it can be inferred that the size of the classes generated by UORB are the smallest. Although the difference from UORB and Madison is not large, it should be noticed that UORB classes are not compressed. In this regard, the support of UORB for legacy systems can be considered more appropriate for legacy embedded systems where physical space is the major concern.

Another point that should be stressed is that UORB, as well as Madison, do not differ much from the client and server set of classes. This feature derives from the fact that the core client and server functionalities of the middleware concerns of the device are centralized in the surrogate object. Therefore, the communication protocol between the device and the surrogate does not vary considerably whether the device requests or offers a determined service.

The same feature does not happen to Orbix/E for the reason that the device implements itself the interface of a CORBA client or server. However, the load balancing in this case is more distributed than that of UORB and Madison cases and thus, the more the devices are connected to the UORB Surrogate Host, the more the performance degrades. In order to overcome this problem in UORB and Madison, a scalable solution should be adopted (see The Benefits of a CORBA Surrogate section regarding the Surrogate Hosts Federation).

5.3.2 Memory Resource

Although the scenario was a simulated, some assumptions of the processing power required from the device could be made.

Comparing the Surrogate approach with the MinimumCORBA, it can be assured that the processing power of the device required in the first approach is less than that of the second. This difference comes from the fact that the middleware concerns of the Surrogate approach are centralized in the Surrogate Host, while the MinimumCORBA approach shares the device processing with other CORBA hosts.

All in all, in terms of RAM, the memory resource required in the three approaches is fairly small, but this depends on the number of objects in the system, the operating system and, in the case of Java classes, the virtual machine adopted.

5.3.3 Qualitative Concerns

The other trade-off that deserves consideration relates to the qualitative aspects deriving from the Surrogate and the MinimumCORBA models. Some concerns such as modularity, reusability level and adaptability are the principles of software quality that show up when considering both models.

Since the MinimumCORBA does not provide a programming model, it should be mentioned that the qualitative concerns that MinimumCORBA can support will vary primarily on the ability of the designer of the application to be developed.

Conversely, the Surrogate model naturally inherits quality of software principles. The separation of the Surrogate Architecture in logical parts, incurred in package units, along with the adoption of some design patterns (see UORB Implementation section) sorts out the classes in such a way that further contributions can be supported relatively easy, depending on the kind of modification desired. For instance, the division of the Surrogate model into Surrogate Host and Adapter brings modularity by separating programming complexities in two major parts where network protocol concerns may be dealt with in the Adapter module and middleware concerns are concentrated in the Surrogate Host.

In regard of reusability, it can be brought up the separation of the device primitives from that of the surrogate. Any modifications of the CORBA resources the device uses are concentrated on the surrogate classes requiring, if desired, slight modifications on the device. For example, acting as a client, if a device wants to make use of an additional CORBA service, the modifications can come off only in the surrogate class. Therefore, it may be transparent to the device the modifications realized.

Finally, in terms of adaptability, the original support for multiple interconnect adapters is what actually characterizes this software principle. Regarding the Jini Surrogate model, there are ongoing projects [17] in the Sun community aiming at interconnect supports other than IP networks. Instances of these projects include the IEEE 1394, USB (Universal Serial Bus) and Bluetooth.

Regarding UORB, the only Interconnect Adapter implemented so far is the IP Adapter. In order to support other Interconnect Adapters, modifications would be required, i.e. the new Interconnect Adapter would have to extend the Adapter interface. It could be realized by using the CORBA inheritance or delegation facility.

6. Conclusions

This work's main contribution is to provide a communication mechanism between devices and parts of a distributed network where devices are not required to embed a middleware profile. As the CORBA Surrogate model achieves this goal, devices are, therefore, loosely coupled to CORBA. The accomplishment of this target was developed throughout this paper.

By means of UORB, devices become part of a network of CORBA technology-enabled services, devices or other distributed networks through interoperability facilities derived from CORBA. We reasoned why CORBA showed to be appropriate to implement a Surrogate model. In order to properly support the idea, some features that a CORBA Surrogate model can contribute were identified. Some advantages arise from this model:

- It is not restricted to one programming language support for the CORBA independence of language and platform. The device can implement its interface in C, C++ or any other language that CORBA supports and communicate with the CORBA Surrogate Host without restrictions. The Jini Surrogate Model, on the other hand, is tightly coupled with Java.
- Considering that the surrogate object is a CORBA object that acts on behalf of the device, it can fully comply with CORBA principles. The device, by means of the surrogate, may then make use of CORBA Services, even the Dynamic Interfaces of CORBA. The MinimumCORBA does not provide support for Dynamic CORBA Interfaces.

Through the IP Interconnect protocol, the purpose of ubiquity stated by our CORBA Surrogate model can be carried out. In other words, that protocol fulfills the discovery mechanism, which provides flexibility of the device to plug in the network whenever it wants. Whenever there is a Surrogate Host available to receive incoming device requests, the Surrogate object standing for the device is registered in the Surrogate Host, enabling the Surrogate to be loaded and activated. Additionally, this feature ensures that changes can be made to the Surrogate object and that the correct drivers are used the next time the device intends to participate within the CORBA network.

Additionally, some benefits can be drawn from the comparison between UORB, Madison and the MinimumCORBA implementation:

- Although the programming of the classes to embed may be harder, it is more efficient for it does not have a layer bottleneck stemmed from the middleware. In fact, the implemented classes from one device to another may be quite similar, since they use the same interconnect adapter.
- The size of the classes to embed in the device is so short that not only current devices can make use but also legacy devices that intend to communicate with a CORBA network.

6.1 Directions for Future Work

A future contribution that would give more flexibility to the current model would be the use of the facility of the CORBA Dynamic Interfaces. In other words, the device would only provide its interface and the CORBA Surrogate Host would interpret this interface and construct the CORBA object through the CORBA DSI or the CORBA DII.

Another useful contribution would be the adaptation of UORB to scalability. As presented in The Benefits of a CORBA Surrogate section, the construction of a federation would be interesting. With this contribution, a Surrogate Host Federation can be conceived in order to meet performance, scalability, availability and fault-tolerance requirements. The JTF pattern in [1] would be appropriate to support this idea.

References and Bibliography

[1] M. D'Amorim. *Service Trading on the Internet, the JTrader Approach*. MSc Dissertation Thesis, Federal University of Pernambuco. May, 2001.
[2] E. Gamma, et al. *Design Patterns – Elements of Reusable Object-Oriented Software* – Addison-Wesley, 1995.
[3] R. Klefstad, D. Schmidt et al. *Towards Highly Configurable Real-Time Object Request Brokers,* in International Symposium on Object-Oriented Real-Time Distributed Computing 2002, April 29 - May 1, 2000.
[4] H. Kopetz. *Design Principles for Distributed Embedded Applications*. kluwer Academic Publishers, 1997.
[5] C. O'Ryan, D. Schimidt, et al. *Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0.* IEEE RTAS' 2000, 31 May – 2 June 2000.
[6] Object Management Group. *CORBAServices Specification*, October 1998.
[7] Object Management Group. *The Common Object Request Broker: Architecture and Specification* – Revision 2.5, September 2001.
[8] Object Management Group. *MinimumCORBA Specification*, August 1998.
[9] K. Raza, et al. *Plug-and-Play Network Service Configuration Using CORBA*, in Fifth IFIP Conference on Intelligence in Networks, November 1999.
[10] M. Roman, et al. *Integrating PDAs into Distributed Systems: 2k and PalmORB*, in International Symposium on Handhelds and Ubiquitous Computing (HUC 99). Germany, September 1999.
[11] M. Roman, et al. *LegORB and Ubiquitous CORBA*, in Reflective Middleware Workshop IFIP/ACM, April 2000
[12] Sun Microsytems. *CLDC and the K Virtual Machine (KVM)*, January 2001.
[13] Sun Microsystems. *Jini Technology Core Platform Specification*, 1.1 edition, October 2000.
[14] Sun Microsystems. *Jini Device Architecture Specification,* October 2000.
[15] Sun Microsystems. *Jini Technology IP Interconnect Specification*, August 2001.
[16] Sun Microsytems. *Jini Technology Surrogate Architecture Specification,* July 2001.
[17] Sun Microsystems. *Surrogate Architecture Interconnect Projects*. Available at http://surrogate.jini.org/rellinks.html