

CORBA Application in Real-Time Distributed Embedded Systems

Yanfeng Gong

Abstract—This paper reports CORBA applications in modern real-time embedded systems design. Important features of CORBA, Minimum CORBA, and Real-time CORBA will be briefly reviewed. Key characteristics of contemporary real-time embedded systems will be covered as well.

Key Words—CORBA, Real Time, Embedded Systems, Distributed Computation.

I. INTRODUCTION OF CORBA

The explosive growth of information, the increasing diversity of information systems and the growing popularity of high-speed network connection challenge enterprise system integration in several aspects:

- Different system platforms and programming languages
- Coexistence of client-server or mainframe oriented system application
- Lack of a well-defined architecture.
- Conflicting data formats and semantic definitions

In recognition of these challenges, the Object Management Group (OMG) eventually defined the CORBA, acronym of Common Object Request Broker Architecture, standard. CORBA is a vendor-independent specification designed to let different platforms - CPU, operating system and compiler - work together seamlessly. The CORBA specification is supported by more than 700 (and growing) hardware and software manufactures, government organizations, and user groups. It has also been implemented by numerous hardware and software system manufactures, creating a rich and robust framework that successfully operates across heterogeneous computing platforms.

CORBA is solidly grounded in fundamental object-oriented programming and is based on a client-server model of distributed computing. The client of a CORBA object acquires its object reference and uses it as a handle to make method call, as if the object is located in client's own address space. The Object Request Broker (ORB) is responsible for all the mechanisms required to find the object's implementation, prepare it to receive the request, communicate the request to it, and carry the replay (if any) back to the client. The core of

the CORBA architecture is the ORB that act as the object bus over which objects transparently interact with other objects located locally or remotely [1]. Not only does the broker provide common services, including basic messaging and communication between client and service, It also insulates the application from the specifics of the system configuration, such as hardware platforms and operating systems, network protocols, and implementation language. Figure 1 [2] shows the structure of a typical ORB. The interface to the ORB is shown in strip boxes, and the arrows indicate whether the ORB is called or perform an upper-call across the interface.

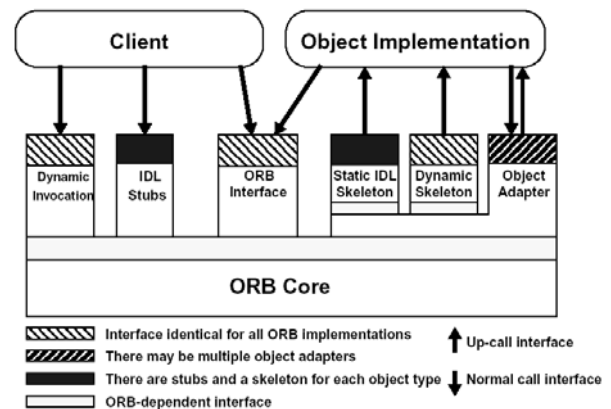


Figure 1 The Structure of Object Request Interface

To invoke operations on a remote distributed object, a client must know the interface offered by the object. The interface, composed of operations and the types of data that required to be passed to and from those operations, is defined in the OMG Interface Definition Language (IDL). Unlike other programming languages, such as C++ and Java, IDL is a declarative language to define object interface in a manner that is independent of any particular programming language. With different available IDL compilers, IDL can be translated to different programming languages, such as C, C++, Java, Smalltalk and Ada. The standard languages mapping make it feasible that developer can implement different portions of a distributed system in different language. For example, based on CORBA technology, a high-throughput server application may be written in C++ for efficiency and client can be developed using Java Applet. The language independence of CORBA is key to its value as an integration technology for heterogeneous systems [3].

There are two general approaches specified by OMG through which client invokes request and server receives request

1) *Static invocation and dispatch*

In this approach, every CORBA object must be defined by IDL that is then translated into language-specific stubs and skeletons by compiler. A Stub is a client-side function that enables a request invocation to be made in a local function call fashion. Similar, a skeleton is a server-side function that enables the object resident server to receive distant method invocation. Because of static compiling, both client and server know the programming language types and functions mapped from the IDL description of remote objects.

2) *Dynamic invocation and dispatch*

In this approach the CORBA object has to be defined by IDL in server side, but it is not necessary to be translated. Client requests invocation directly through Dynamic Invocation Interface (DII) without the compile-time knowledge of the implementation of the object. Similarly, server receives and dispatches remote method invocation from client through Dynamic Skeleton Interface (DSI).

Object adapter is the primary mean for an object implementation to access ORB services such as object reference generation, and serves as the “glue” between servants and the ORB. The object adapter defines most of the services from the ORB that object implementation can depend on.

In order to enable various ORBs from different vendors to seamlessly communicate to each other, OMG specified a general ORB interoperability architecture named General Inter-ORB Protocol (GIOP). GIOP is an abstract protocol that specifies transfer syntax and a standard set of message formats of message request ORBs can make over different network. The GIOP implementation over TCP/IP is called Internet Inter-ORB Protocol (IIOP) by that ORBs can effectively use Internet as an ORB communication bus. Figure 2 illustrates the usage of IIOP. Since the IP is the only broadly deployed network layer protocol, IIOP is currently the only standard mapping of GIOP.

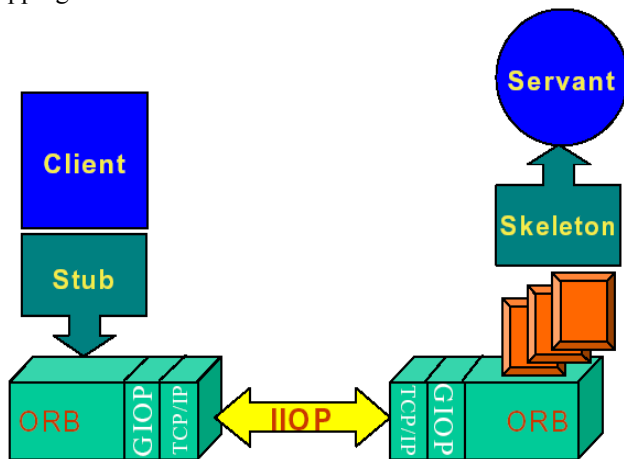


Figure 2 GIOP and IIOP

II. CONTEMPORARY EMBEDDED SYSTEMS

Embedded systems, small and specialized computer systems including hardware and software, are specifically designed for particular kind of application devices, different from general-purpose computer. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. In order to reduce the product cost, general embedded systems have relatively limited resources (hardware and software) to utilize, such as small memory storage or no operating system. Even with decreasing cost of microprocessor and peripheral hardware due to the technology advance in semiconductor industry and some free embedded operating systems available, cost still imposes the biggest challenge on embedded system design. A typical traditional embedded system is an independent but complete system provides several functions (Figure 3)

- 1) *Measurement and monitor*, embedded system reads data from input sensor, and then the data is processed and displayed in some format to user.
- 2) *Control*, embedded system generates command for actuator based on interactive system input or preprogrammed instructions
- 3) *Information process*, such as data compression or decompression.

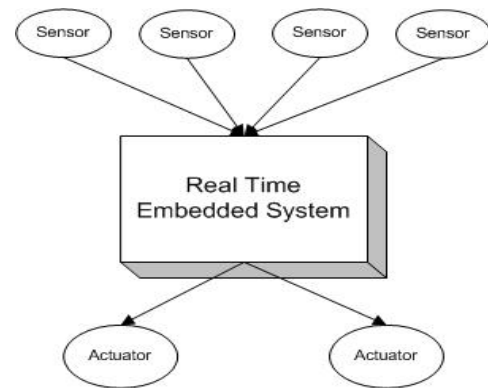


Figure 3 Sensors and Actuators of an Embedded System

With advance in embedded system computation ability and Real-Time Operating System (RTOS) available, more and more embedded systems are required to react to event or complete task in specific time. These systems are called real-time embedded systems. Usually, time constrains satisfaction is achieved by the operating system. Real time system can be further divided to two categories:

4) *Hard real-time system*

The classic definition is that hard real-time applications fail if they system timing requirement is not met. Good examples are digital fly-by-wire system of aircraft and missile self cruise system.

5) *Soft real-time system*

The Soft real-time applications can tolerate some degree of

latency in what they are required by the system. The time constrains violation is not fatal, but the system performance may degrade. Examples are vendor machine and printer controller.

The increasing customer demand for distributed applications, coupled with advances in the techniques of networking hardware and high-speed process, has made distribution integration mandatory in many embedded system design or system upgrade. Many modern embedded systems hardware consists of a mixture of general-purpose processors (GPP), digital signal processors (DSP), and network processors (NP). Heterogeneous operating systems also coexist in distributed embedded systems. Therefore, there are many key design issues need to be considered for a contemporary distribution embedded infrastructure.

1) *Distribution flexibility*

The distribution infrastructure should support location transparency, where the physical location of the target object should not be designer's concern. When communication is remote, designers should be isolated from the details of finding objects, using the underlying network to communicate with, and so on.

2) *Heterogeneous systems*

Seamlessly integrate heterogeneous systems (different programming language, operating systems, and processor types) requires standard distribution middleware to enable communication between objects insensitive to the programming language, data representation, communication channel, and invocation techniques.

3) *Distributed real-time constrains*

In distributed real-time embedded systems, the communication time delay of remote method invocation has to be considered in real-time constrains. The underlying choice of transport mechanism (protocols and networking hardware) has a significant impact on overall system performance.

4) *Memory limitation*

Individual embedded system still has memory limitation, so a small memory footprint is essential for any installed application software.

III. CORBA APPLICATION IN EMBEDDED SYSTEM

Enabling and achieving the seamlessly integration and interoperability among different types of embedded systems, even with personal computers (PC), requires the ability to share application interface definitions and to communicate through heterogeneous environments. This requirement has also being existed in the enterprise system integration of different business subsystems, such as manufacturing systems, inventory control, accounting and marketing systems.

CORBA has firmly established itself as the industry standard for distributed computing in enterprise systems world and has been well accepted by my industries as the preferred middleware solution by a large number of industries. CORBA has been proven to provide almost universal interoperability with general-purpose computing platforms and easy

integration with existing applications. Therefore, the adoption of CORBA in the real-time embedded systems arena naturally. Although it is ideally possible to build embedded system application that contains matured enterprise CORBA technology, it did not take long for embedded system designer to realize that enterprise CORBA technology has to be optimized to fit real-time embedded systems' unique requirements. The two important issues that had to be addressed were support for embedded systems' memory limitation and the relationship between CORBA and preemptive multitasking real-time operating systems on which it executes. Recognizing these two issues, and to facilitate CORBA's adopting, the OMG has made two subsets particular for its application in embedded real-time embedded systems: Real-time CORBA and Minimum CORBA.

A. *Minimum CORBA*

Minimum CORBA, a sub set of CORBA, provides guidelines to develop a "minimal" CORBA to meet resource-constrained system requirements without compromising the full interoperability between Minimum CORBA and full CORBA applications.

Bearing the same goals, "portability" and "interoperability", as CORBA, the Minimum CORBA specification fully supports all of OMG IDL, defined in CORBA specification. Which features are retained in Minimum CORBA and which are stripped out has been carefully chosen by OMG Minimum CORBA group to yield a profile that still has broad applicability within the world of limited resource systems, such as embedded systems. The omission of the features of CORBA represents a trade-off between usability and conserving resource [4]. Features omitted from CORBA could still be implemented in applications as long as they are needed and system resource permits. For example, because embedded systems almost always have their functionality known at compile time, the features of dynamic interface definition and dynamic invocation are removed by Minimum CORBA. For more details about which dynamic related function is omitted, please refer to [4].

With the latest ORBs on the market, Minimum CORBA can really work for resource constrained system. For example, *ORBexpress* CORBA has a static memory footprint of less than 100k on some platform, while still keeping most of the functionality of full-sized CORBA implementations [4].

B. *Real-time CORBA*

Real-time CORBA is an optional set of extensions to CORBA tailored to equip ORBs to be used as a component of a real-time system [5]. The current version of real-time CORBA is sufficiently able to cover both hard real-time and soft real-time requirements. But it only supports fixed priority scheduling. Dynamic scheduling will be addressed in later version.

The goals of Real-time CORBA specification is to support developer in building predictable distributed system, a prerequisite for ensuring real-time performance, by providing

mechanisms to control the use of processor, memory, and network resources. The application manages the resource by using the Real-time CORBA interfaces and the ORB's mechanisms to coordinate the activities that comprise the application. The Real-time CORBA also relies upon the RTOS to schedule the threads that represent the activity that being preceded and to provide the mutexes to handle resource contention [5]. Specifically:

1) Processor Resource Management

In order to make processor to be used in a predictable way, Real-time CORBA assigns "Real-time CORBA Priorities" to operations of CORBA Objects. These priorities are consistently mapped down to the priorities of underlying RTOS task/threads and allow the scheduling of the CORBA operations to be integrated with the scheduling of non-CORBA parts of a system, producing a single, consistently schedulable system. As shown in Figure 4 an ORB endsystem [6] consists of network interfaces, operating system I/O subsystems and communication protocols, and CORBA-compliant middleware components and services.

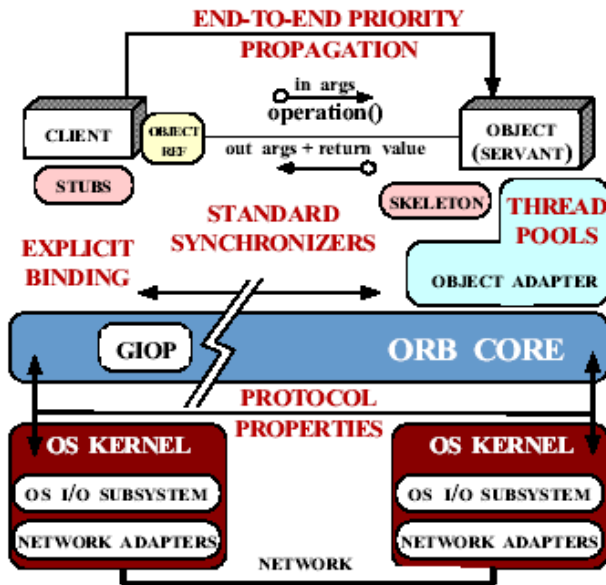


Figure 4 ORB Endsysteem Features for Real-Time CORBA

The range of native priorities that is used in the mapping might be different for each OS. Figure 5 illustrates two different ways to map the Real-time priorities to native OS priorities. The priorities can be mapped either onto the entire native priority range (as shown in the case of OS #1), or onto a sub-set of the native range (as shown in the case of OS #2.)

2) Memory Management

Real-Time CORBA allows memory resources to be used in a predictable fashion through the control over threadpools. This allows the number of threads that are used by a CORBA subsystem and the amount of memory each thread takes up to be controlled. The amount of memory reserved for the queuing of CORBA requests can also be configured, through a 'request buffer' parameter on each threadpool.

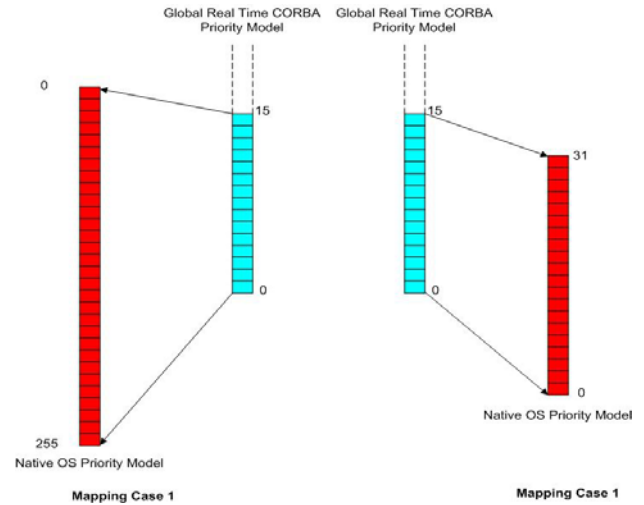


Figure 5 Real-time CORBA Priority Mapping to native OS priority

3) Network Resource Management

There are three ways can be used by Real-time CORBA to allow network resources to be used in a more predictable way.

a) The application can select between and configure the available network protocols. To date, only TCP/IP has been adopted by OMG. Since many real-time systems require either higher performance or greater predictability from the network transportation, some CORBA vendors have developed or implanted different protocols within their CORBA implementations.

b) Real-Time CORBA allows a client to obtain a private transport connection to a server, which will not be multiplexed (shared) with other client-server object connections.

c) Real-time CORBA provides the facility for a client to communicate with a server via multiple connections. The selection of the appropriate connection is transparent to the application, which uses a single object reference as normal.

For more Real-time CORBA IDL specific information, please refer [5].

Because system timeliness is decided by many other issues besides ORA applications, real-time CORBA compliant distribute system will not guarantee that system deadlines will always be met. But it will at least assure the system developer that the middleware layer will behave in a deterministic manner.

IV. CONCLUSIONS

Embedded systems are becoming more distributed, real-time critical, and heterogeneous. Since CORBA has been successfully dealing with same problems in enterprise computer arena, some extensions have been made to make CORBA as a good solution to embedded systems as well. Leveraging the high developed, technically robust, and widely compatible CORBA technology, distributed embedded system developers will be able to significantly reduce design and

maintenance costs and be free of worrying about system compatibility.

REFERENCES

- [1] S. Vinoski, *CORBA: Integrating diverse applications within distributed heterogeneous environments*, IEEE Communications, vol. 14, No. 2, Feb. 1997
- [2] Common Object Request Broker Architecture (CORBA/IIOP) Specification, Version 3.0.2. The Object Management Group, December 2002. Available electronically at http://www.omg.org/technology/documents/corba_spec_catalog.htm
- [3] Michi Henning, *Advanced CORBA Programming With C++*, Addison-Wesley, 1999
- [4] Minimum CORBA, Version 1.0. The Object Management Group, August 2002. Available electronically at <http://cgi.omg.org/docs/formal/02-08-01.pdf>
- [5] Cathy Hrustich, "CORBA For Real-Time, High Performance and Embedded Systems", Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 200. pp. 345-349
- [6] Real-time CORBA, Version 1.1. The Object Management Group, August 2002. Available electronically at <http://cgi.omg.org/docs/formal/02-08-02.pdf>
- [7] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp294-324, Apr. 1998