

Why Multiprocessor DSP Systems Need CORBA

CORBA enables software components in a multiprocessor system to easily communicate—regardless of what language they are written in, what OS they run on, or where they are located. Even better, CORBA makes it easy to move functionality between DSPs, GPPs, and FPGAs.

By Joseph M. Jacob

Signal processing systems often include multiple types of processors, such as Digital Signal Processors (DSPs), General Purpose Processors (GPPs), and Field Programmable Gate Arrays (FPGAs). These disparate processors must interoperate with one another, which presents several challenges.

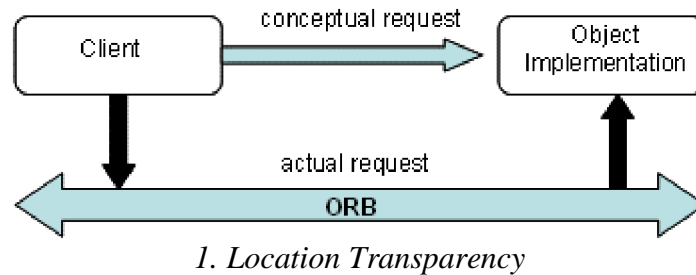
In larger development projects, DSP programming groups often operate in a separate silo from the GPP and FPGA programmers. This is a problem because the groups must coordinate their efforts in order to design an optimal system architecture and to allocate functionality optimally. This coordination can be time-consuming and challenging. What's more, any changes to the system architecture or functional allocation can result in substantial code re-writes for each programming group, greatly slowing time-to-market.

These challenges, important as they are, are not unique to DSP systems. Many other industries must integrate disparate platforms into a single system. To meet this goal, these industries rely on a distributed communication architecture known as the Common Object Request Broker Architecture (CORBA). Thanks to recent developments, programmers can now take advantage of this technology in real-time signal-processing systems.

What is CORBA?

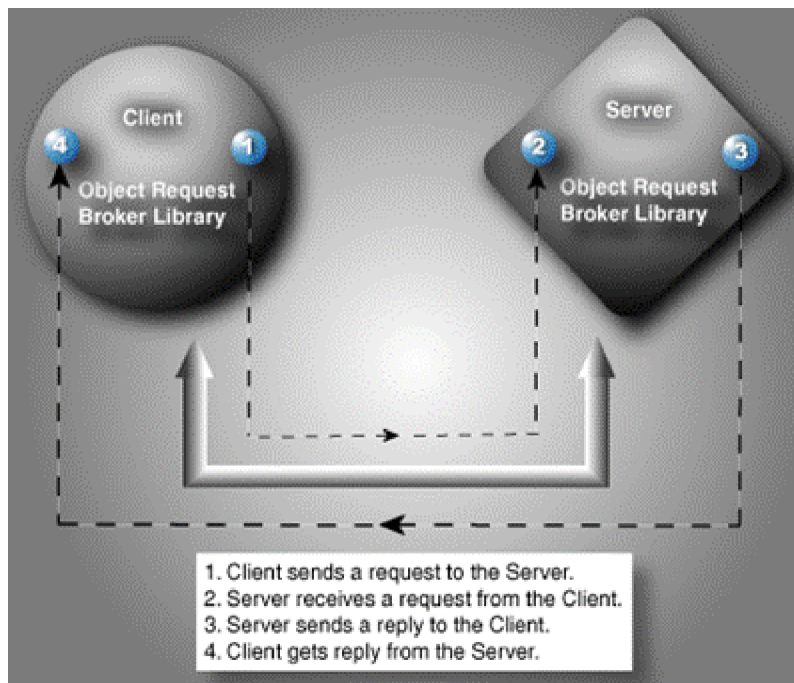
[CORBA](#) is an open, vendor-neutral standard created by the Object Management Group (OMG) consortium. CORBA enables pieces of programs, called objects, to communicate with one another over networks—regardless of what [programming language](#) they are written in, what [operating system](#) they are running on, or where they are located in the system.

CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed. The basic idea behind CORBA is the concept of location transparency: It makes no difference whether an object is called within the same processor or on a remote processor. This concept is illustrated in *Figure 1*.



CORBA uses a [client-server](#) model to manage communications between objects. The client/server interfaces are defined using the Interface Definition Language ([IDL](#)). The IDL is language-neutral and platform-independent: IDL definitions can be mapped into any popular programming language, such as C or Java. This allows objects to communicate regardless of the language and platform underlying the objects.

Data communication from client to server is accomplished through a well-defined object-oriented interface. The Object Request Broker ([ORB](#)) determines the location of the target object, sends a request to that object, and returns any response back to the caller. Through this object-oriented technology, developers can take advantage of features such as [inheritance](#), [encapsulation](#), [polymorphism](#), and runtime dynamic [binding](#). These features allow applications to be changed, modified and re-used with minimal changes to the original object. The illustration below identifies how a client sends a request to a server through the ORB:



2. Illustration of client sending a request to a server.

There is much mystique in the software development community around "object-oriented development." But the reality is much more mundane and straightforward. Objects package both data (called properties) and the procedures (called methods) that operate on

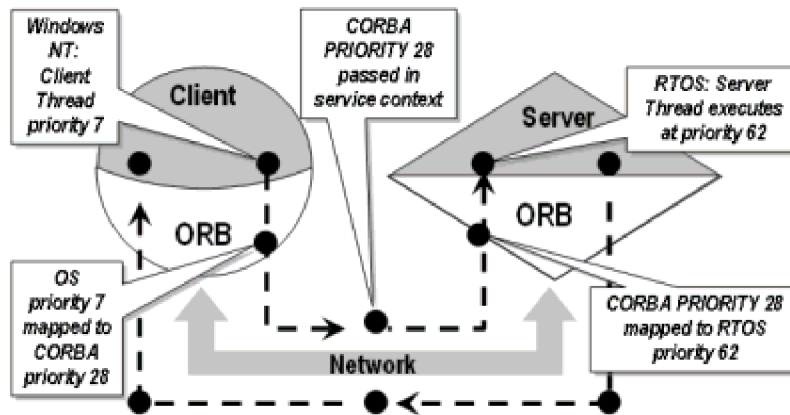
that data. As a result, these packages become convenient units of processing for the purposes of distribution and reuse. Object Oriented Design encourages the definition of objects based on the application problem, e.g., "power spectrum" rather than solution, e.g., [DFT](#). An object-oriented approach is ideal for homogenous, multi-processor systems, as it allows developers to focus on the desired functionality rather than the details of the underlying hardware.

Real-Time CORBA

Real-time CORBA adds deterministic behavior—its primary goal is achieving end-to-end predictability in a distributed system. Predictability is relatively easy to achieve on a single-processor system through use of priority-base scheduling. Predictability is harder to achieve in a multiprocessor, networked system because the processors do not share information about the priorities of their tasks. Real-time Object Request Brokers (ORBs) solve this problem by propagating the priority information. The result is distributed priority inheritance and the ability to schedule distributed processes to achieve predictability.

The heterogeneous nature of CORBA is important in priority propagation. Consider the fact that every Real-Time Operating System (RTOS) has its own range of priorities. In one RTOS, the priority range may be 0 to 255, where 255 is the highest precedence. Another RTOS may have a range from 0 to 63, where 0 is the highest precedence. Manually coordinating priorities across different operating systems is challenging because your code has to know which operating systems you're using. The Real-time CORBA standard simplifies matters by defining a universal priority range—just one way CORBA provides a portable and transparent approach to building distributed systems.

Real-Time CORBA introduces a range of Real-Time CORBA Priorities. Real-Time CORBA Priority maps into native operating system priority throughout the system in a consistent manner, and specifies how to reconcile communication requests from the client application, through the middleware layer, over the network, and up to the server application without unexpected priority inversions. When a client invokes an operation to a server, the client's native operating system priority is mapped to a Real-Time CORBA Priority, which is carried as part of the message to the server. The following illustration shows the real-time priority mapping function of the client/server application, through a CORBA implementation.



[\(Click to enlarge\)](#)

3. End-to-End Predictability for the Activity

When the Server receives a message, the server-side ORB maps the priority to the server's operating system native priority level, and invokes the operation on the server at the same relative priority level as the client. The result is distributed priority inheritance, providing end-to-end predictability in the same manner that would be expected for an operation in a single process.

Priority propagation ensures that the RTOSes can be scheduled consistent with overall priority. However, the network can still be a source of priority inversions. A high-priority message may have to await the completion of a large low-priority message. That represents a message-based priority inversion. To avoid this bottleneck, Real-time CORBA supports Priority-Banded Connections. Priority banded connections allow the software developer to separate different types of traffic by priority, and to dedicate separate connections between the client and server for each priority "band", where a band can encompass any set of CORBA Priority levels. This allows greater predictability for traffic, and allows for a mix of real-time and non real-time data on the communications transport. Priority banding also helps to bound and minimize the chances for any priority inversions.

(More information on Real-Time CORBA is available at <http://www.ois.com/resources/corb-3.asp>. For more information on CORBA in general, see <http://www.ois.com/resources/corb-2.asp>.) **What is CORBA/e?**

For systems that need small memory footprint and deterministic execution, DSP programmers can use the latest generation of CORBA: CORBA/e (CORBA for embedded). An Object Management Group (OMG) standard, CORBA/e provides an architecture for distributed processing that fits systems from the largest server farms to the smallest networked DSPs.

The CORBA/e Compact Profile fits easily on a typical 32-bit microprocessor running a standard Real-Time Operating System (RTOS). The CORBA/e Micro Profile is even smaller and fits on the kind of low-powered microprocessor or DSP found on mobile or hand-held equipment. As with standard CORBA, CORBA/e provides the benefit of code

reuse. Developers do not need to re-write code for all systems on a network when they want to make changes, thus preserving their investment in existing applications.

Why CORBA/e for DSPs?

CORBA/e enables DSPs to become first-class devices in mixed systems. This means that the portability of the CORBA interface provided to DSPs lets DSP developers and the system architect optimize functionality in the DSP with the same ease they now have in the GPP. CORBA/e is ideally suited to the challenges of today's mission-critical environments:

Standalone systems are a thing of the past. DSPs need to communicate and interoperate with GPPs, FPGAs and other DSPs. The systems in which they operate are networked and highly interconnected. Software must cope with communications and interoperability issues, while delivering the same reliability and performance as the isolated embedded systems of the past. Even systems that appear to be standalone often need a communications infrastructure to merely report their status to a central control system. CORBA/e makes this interoperation manageable. What's more, it provides easy access to a variety of sophisticated transports tuned to embedded targets including Shared Memory, PCI, PCI Express, Rapid IO, Firewire, and Ethernet. CORBA/e also allows DSP programmers to "plug-in" their own custom transports.

Platform flexibility is essential. It was once acceptable to target embedded software to a specific processor on a particular board. Today, there is increased pressure to preserve investment through development of reusable code that can be used with different targets. DSP programmers may need to target many different processors at the same time, and they may need or migrate to new compilers, operating systems and processors over time. Without an appropriate infrastructure, these requirements make a project susceptible to repeated code changes. CORBA/e insulates DSP developers from the headaches of rewriting code with every processor and system change.

Multicore Processors. Embedded systems increasingly use multicore DSP processors. Developers currently writing code for single-core systems will need to migrate their applications from single core processors to processors containing two or more DSP cores. CORBA/e enables a seamless migration to multiple cores through the benefits of location transparency.

Embedded systems interact with the real world in real-time. Devices with DSPs require interactions that are predictable in time as well as in function. CORBA/e provides distributed predictability by recognizing and propagating priority in its own processing and across the system.

Power, weight, size and speed are constrained. CORBA/e is specifically designed to support board-based and networked systems with the smallest footprint and the highest performance requirements.

Reliability must be built-in. DSP programmers are rightly skeptical about adopting code they don't write themselves. Robust implementations of CORBA/e like [ORBexpress RT](#) and [ORBexpress DSP](#) have been field-tested in the toughest environments, proving their reliability time and time again.

Application flexibility through refactoring. DSP programmers and their GPP programmer counterparts can build a CORBA/e application as if it were a standalone application. They can then distribute the application across multiple resources with minimal effort. The programmers can also change the distribution of the application with ease, making it possible to defer deployment decisions until late in the design process. Deferring deployment decisions enables optimized resource allocation and the flexibility to adjust to changing conditions. Most importantly, it allows DSP programmers to easily move unnecessary algorithms off of a DSP to a GPP or FPGA in order to ensure that the DSP performs only those tasks for which it is optimized.

Time to market is critical. Economic pressures are requiring greater productivity from systems development. By supplying a high-performance communications framework, CORBA/e enables greater productivity because it is no longer necessary for DSP programmers to write their own communications protocols and to re-write those communications protocols every time there is a change in software or hardware architecture. By providing a reliable, flexible architecture, CORBA/e eliminates one of the most tedious and time-consuming parts of distributed application development. This ensures that a system architect can move algorithms among a DSP and a GPP to achieve greater total system efficiency without incurring any significant code or communications protocol rewriting—even if these moves come very late in the development process.

Conclusion

Embedded systems are called upon to interoperate in many ways: An automobile, a circuit-board assembly unit, or even a sophisticated office copy machine may contain multiple embedded DSPs and GPPs, connected by a network. In an assembly plant or chemical refinery, process controllers may interoperate with many small sensor units, and one or a few large servers or mainframes.

When embedded in automobiles, airplanes, weapons systems, hand-held radios and cellular telephones, and other devices, software must work as reliably as the hardware. This is challenging because the typical embedded DSP environment is networked, forcing its software to deal with communications and interoperability issues without compromising reliability and performance.

The interoperability and dependability of networked embedded applications can only come from a proven, standard middleware. The new availability of CORBA/e-based ORBs for DSPs is exactly what is needed for these applications.

CORBA/e offers an architectural solution for DSP programmers and embedded system architects to keep up with the rapid pace of technological change " in processors, models,

and particularly communications bus types. CORBA/e lets developers protect their investment in development work despite rapidly accelerating change.

The benefits of CORBA/e for DSPs may be summed up this way:

- A proven, high-performance architecture used in the most demanding environments.
- DSPs become first-class devices in system processing.
- High-performance and speed within a small footprint.
- Reduced risk—changes can be made late in the software development process without rewriting code, while preserving existing investments in application development.

About the author

Joseph M. Jacob is the Senior Vice President, Sales and Marketing of Objective Systems, Inc. Since joining Objective Interface in 2003, Joe has led the company's sales, marketing, business development and product management teams. Prior to joining Objective Interface, Joe's professional experience included leading the international business development and strategy function for America On Line. During his career, Joe has worked and lived throughout Europe, including England, France and the Czech Republic. Joe holds a B.A. from the University of Illinois at Urbana-Champaign with a double major in economics and political science. He also holds a Juris Doctor degree from Harvard Law School.