Factors to Consider When Selecting CORBA Implementations

Dr. Thomas J. Croak
Computer Sciences Corporation

Performance and flexibility are old rivals in computer architecture. Common Object Request Broker Architecture (CORBA) certainly provides flexibility, but at what cost to performance? This article summarizes middleware research resulting in the identification of "10 dimensions of variability" in software architecture that can cause each implementation of CORBA to behave differently in a given system. The resulting factors can be used to choose a CORBA Object Request Broker implementation, and then tune it to the specific architecture.

All distributed software architectures require a mechanism to exchange data between the nodes of the architecture. A generic term for this class of software is *middleware*. Common Object Request Broker Architecture (CORBA) is an open standard for object-oriented middleware developed by the Object Management Group, a consortium of nearly 800 companies. A CORBA-compliant Object Request Broker (ORB) can facilitate data communication between dissimilar hardware, dissimilar operating systems, and modules written in different languages. CORBA is a Joint Technical Architecture (JTA) endorsed standard, and is a part of the Defense Information Infrastructure (DII) Common Operating Environment (COE).

Command and control (C2) systems must be able to provide decision support information in response to the battle as it is taking place and therefore must be very efficient. The ability of a system to meet performance requirements depends heavily on the underlying architecture selected for the software system. Middleware is a critical performance component. Most C2 systems being built to JTA and DII COE standards have CORBA as their middleware. There are many choices of CORBA ORB implementations, and they can each affect performance differently. The basic question then is how does the architect or designer select the best implementation of a CORBA ORB given the performance needs of the system?

This article summarizes middleware research resulting in the identification of 10 *dimensions of variability* in software architecture that can affect a CORBA ORB's behavior for a given system [1]. The results of this research can aid in determining what factors of a specific architecture affect performance when using CORBA. These factors include aspects of the selected hardware architecture that would favor one CORBA implementation over another, and details about how CORBA works that affect the performance of the architecture. The understanding of these factors can help quantify the needs of the architecture, and in turn help evaluate candidate implementations of CORBA.

A secondary benefit of this research is that knowledge of these factors can be used to tune existing systems or systems under construction that have already selected a CORBA product. Most CORBA products have parameters that can be tuned by the system designer to complement the system's architecture and improve performance.

It Is Like Buying a Car

The analogy to this situation is purchasing an automobile. Some people select a vehicle based on features or options without much consideration of how the vehicle will be used. Many CORBA ORBs are purchased the same way. This is possibly because both cars and CORBA ORBs have extensive marketing literature focused on features and options.

A better approach is to perform a needs analysis to determine which vehicle factors are most important, and then compare vehicle choices directly using common units of measure for those factors. The first step of a needs analysis would be to determine what the primary purpose of the vehicle will be. What is secondary? How many people should it hold? What else will be carried in the vehicle? Should it be able to tow a camping trailer or a boat trailer? The next step is to select factors for comparison such as acceleration, gas mileage, towing capacity, and range. There are a series of standard units of measure that can be used to help make this comparison. Acceleration can be compared using the time to go from 0 to 60 mph. Fuel efficiency can be compared using the estimate of mpg. Towing capacity can be expressed in terms of both dead weight and tongue weight. Range is simply the fuel tank size times the mpg estimate.

How can all of these factors be optimized? Basically, it cannot be done. The only possible result would be a vehicle that was not good at anything. The best solution is to determine which factor is the most important to the need and optimize that factor. Fortunately for car buyers, there is a whole series of generally understood units of measure for comparison. The factors to consider when selecting CORBA are not nearly as well understood, and there are few if any generally accepted units of measure for comparison.

Command and Control System Needs Analysis

The first step toward selecting a CORBA implementation should be performing a rudimentary needs analysis to understand the constraints on the design of the system. While a family vehicle must satisfy many different needs, often loosely defined, there is a much better situation for C2 systems. A C2 system is often designed to support a single mission or a closely related group of missions. The requirements for such a system are generally clearly stated in terms that are quantifiable and testable. This statement of need is often referred to in computer science terminology as the *service policy* for the system.

To better understand the service policy for a C2 system, it may be useful to compare it with the service policy for a business system. This is done for the simple reason that most CORBA implementations are actually designed for the business environment. A rudimentary understanding of this environment is important to selecting a CORBA implementation for C2 systems.

A typical service policy at a bank might say: "Ninety-five percent of the transactions must be processed within 10 seconds." Notice that there is no stated requirement for the other 5 percent, and that the time constraint is fairly loose and could be easily met by several architecture

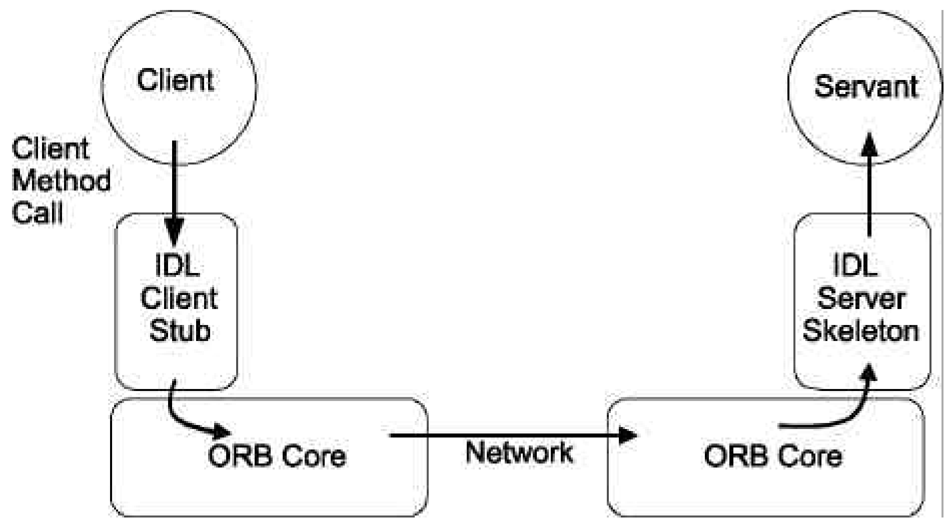

Figure 1: *Generalized Flow of a CORBA Transaction*

configurations. A bank's service policy would most likely be derived from statistical analysis of marketing data. It would show customer arrival rate and how long customers are willing to stand in line, or how long they are willing to stand in front of an ATM while waiting for the transaction to be authorized. Cost per transaction seems to permeate many of the calculations. The bank is looking to reduce the cost per transaction without losing customers.

The bank's service policy might seem very foreign to someone used to working with C2 systems. A service policy for a C2 system might be stated: "All transactions must be processed within 80 milliseconds (msec) while using no more than 50 percent of available processing capacity." This statement is different from the bank's service policy in three important ways. All transactions are covered, not just most of them. The time constraint is far shorter. And perhaps most importantly, a built-in reserve capacity has been stated.

The cost reasonableness for these systems is not based on a per transaction basis, but rather the cost of regrets if the mission fails due to an inefficient computational suite. The advantage the C2 system designer has over the business system designer is that the focus can be on optimizing the system to process a single transaction. That transaction does not even have to be a typical one. It may be the one that is most time-critical or has the highest use of compute power. This is an advantage because the designer can break down that transaction into its component parts and find the processing bottlenecks without having to consider arrival rate statistics. These component parts include the processing steps within the CORBA transaction.

Components of the CORBA ORB Transaction

The basic building block in CORBA is the CORBA object. A CORBA object models a real-world object and consists of the data and the methods that it may invoke. The object's public interface is through the CORBA Interface Definition Language (IDL). The purpose of the IDL is to hide the underlying object's implementation details. Any client of the object can use this interface to invoke the methods of the object without knowing any of the details of the implementation, the platform it is on, or the location of the object [2]. There are IDL implementations for Ada, C++, JAVA, Smalltalk, and several other languages.

The heart of CORBA is the ORB. The ORB acts as the middleware component that implements the conceptual bus between the client and the servant. The ORB ensures delivery of the client's request, while hiding the location and implementation details from the client. Additionally, in its broker role, if there are multiple server components that can perform the method, the ORB performs a load balancing function between them [3]. ORBs are optimized and tested thoroughly resulting in the virtual elimination of many tedious, error-prone aspects of creating and managing distributed applications, while increasing the portability and reusability of the service components [4].

While the ORB is far more complicated than the simple illustration in Figure 1, it contains the principal components involved in the transaction. For more detail and a look at the entire CORBA reference model, check out the Object Management Group's Web site at <www.omg.org>. The general flow of an ORB transaction consists of the following steps. The client object invokes a method call on the servant as if it were performing the call directly to another object. The IDL Stub is the public representation of that method and intercepts the call. The ORB core performs the data conversion functions, the brokering function, and the communications function. The IDL skeleton represents the client to the servant method call.

The CORBA IDL uses the concepts of stubs and skeletons as the glue between the client and servants respectively, and the ORB. Stubs provide either a strongly typed static invocation interface or a more weakly typed dynamic invocation interface, that is used to marshal the client data into the ORB's common packet-level representation of the data. The skeleton does the reverse by taking the packet-level representation and demarshals it back into typed data that is meaningful to the servant.

This stub/skeleton approach hides the complexity of the low-level communication between the client and the server and facilitates the interaction between modules that may have been coded in different languages and between dissimilar hardware representations of the data. An IDL compiler for the language of the component automates the transformation between the CORBA IDL definitions and the target programming language. By performing this automated step, the IDL compiler also greatly reduces the potential for inconsistencies between the client stubs and the server skeletons. Additionally, the compiler eliminates common sources of network programming errors and provides opportunities for automated compiler optimizations [5] and [6].

The CORBA ORB Core provides the brokering role as well as all the communication facilities for sharing resources among processes. The core translates the logical address for the method call presented to it by the client into a physical address and performs all the steps required to ship the data to the corresponding ORB core on the server. That portion of the core performs the other half of the communications protocol and delivers the information to the skeleton. Some ORB cores are optimized to know that if the called servant is on the same computer, many of the communications steps can be skipped.

Dimensions of Variability

The length of time to perform an ORB transaction varies greatly with the implementation of the ORB, how it is tuned, and the architecture it runs on. In order to compare ORB implementations, the architect or designer must first look at the architecture of the C2 system. How distributed is it? How many components are there? How fast are the Local Area Networks and Wide Area Networks? What are the distances traveled? How fast are the servers? How many processors are there? Where are the expected bottlenecks in the architecture?

These are all aspects of the architecture that can affect the performance of an ORB. Various ORBs will also respond differently to various architecture situations. To make an intelligent selection of an ORB, the architect or designer must understand all the factors that will affect the performance of the ORB, and which factors can be tuned to reduce their effect. In addition, the long-term effect of the tuning choices must be considered with regard to the system's maintainability, flexibility, and scalability.

During this research, more than 15 dimensions of variability were identified. Only 10 of these are discussed here. The others such as which language, operating system, and compiler, have less affect and can easily be held constant for the purposes of this evaluation. The remaining 10 will be discussed either independently or in combination. They are discussed roughly in the order that can cause the most impact on the transaction performance. Each is followed by some advice on how that factor can affect the choice of an ORB implementation.

Grain, Bandwidth, and Distance Combined

The first three dimensions of variability are addressed together to show the interplay, then addressed separately to show how they each affect the architecture's performance.

Communication delays are calculated as a function of bandwidth, distance, and data packet size. The reason is that while the distance determines when the first bit arrives at the destination, the bandwidth determines when the nth (last) bit arrives. The architect must choose a grain size that maximizes the combination of the distance and bandwidth. On the surface, it would seem a simple problem that as the bandwidth increases, so should the grain size. It is much more complicated than that.

For example, you want to ship a 1 Mbit file across the United States and receive a reply. At 64 Kbps, about 50 percent faster than a typical home modem, it takes nearly 16 seconds to get the data sent. The 30-msec latency delay for the distance does not add much. Today's standard ATM rate is 622 Mbps (OC-12)[1], with 5 Gbps (OC-96) already in use. At 622 Mbps it takes only 1.6 msec to deliver the message, so the 30 msec latency delay to wait for the reply ends up taking 95 percent of the time. As bandwidth increases, the time-to-reply for this example asymptotically approaches 30 msec [7].

There are other problems that the designer must face in this decision, such as buffer size, efficiency of the computation operation at each end, as well as the time required to checkpoint the data to permanent storage. From this description, it is easy to see why the time to perform the transaction is a function of the object's grain size, the distance traveled, and the bandwidth of the circuit.

"Philosophically, it can easily be seen that any length of chain has a weakest link. Likewise, there is always a bottleneck in any process. One of the key roles of the software system architect is to understand, be able to detect, and manipulate the location of the bottleneck."

Object Grain

Grain refers to the size and layout of the data object being sent to the servant. The size, type of data, and layout of the data can each have an effect on transaction efficiency. Most of the computation time expended during the ORB transaction is applied to the marshalling and de-marshalling of the data. If the data type is a complex record consisting of a mixture of data types, the impact is higher. Large grained data objects also impact the data transport time, especially if the bandwidth is low [6].

So what? If your grain size is large/complex, look for an ORB with efficient marshalling and de-marshalling for the language you intend to use.

Communications Bandwidth

Bandwidth is described in terms of the clocking mechanism of the path and ultimately relates to the bit density of the data to be passed. Bandwidth can vary by several orders of magnitude with a corresponding affect on transaction performance. For a given distance traveled and a given data packet size, the time it takes to deliver that packet is a function of the bandwidth of the communications path.

So what? Take advantage of high bandwidth by finding a CORBA implementation that efficiently handles large data objects.

Distance Traveled

With the advent of giga-bit networks, we have transitioned from being bandwidth capacity constrained to being latency constrained [8]. This fact requires a change in thinking about how much data should be shipped at once. With low-speed networks, the bandwidth drove the decisions on object size. Now the distance traveled can cause the biggest difference in performance. The speed of electrons through a packet-switched network is about two-thirds the speed of light. Use 1msec for every 200 miles as a rule of thumb. If you are doing a method call on a server on the other side of the country, you have 30 msec of latency for the round trip before you add in any of the other performance factors.

So what? For short distances, optimize for an ORB that efficiently handles high volumes of transactions.

Dynamic vs. Static Invocation

Deciding how to invoke the method call also affects the efficiency of the transaction. As stated earlier, static invocation is strongly typed and dynamic invocation is weakly typed. Any Ada programmer knows that strong typing is a good thing and weak typing is not, so this should be an easy choice. However, this line of reasoning ignores one of CORBA's real strengths: the ability to hide the implementation details of the distributed environment from the programmer. Some of those implementation details include in what

[1] *The OC designation stands for Optical Channel, a unit of measure for fiber networks with a single Optical Channel, OC-1, delivering approximately 54 million bits per second.*

programming language the invoked method is coded, and what operating system is running on that server.

Dynamic invocation allows you to make calls for service at runtime without prior knowledge of language or data format requirements of the called service [9]. Static invocation makes little or no changes to the data object prior to shipping it, and requires little or no change prior to presentation to the servant. Obviously, this technique will improve transaction performance, but at the cost of flexibility. Static invocation can be used if the language, compiler, and operating system (and hardware) are known to be the same on both client and server.

So what? If using static invocation, optimize for a CORBA ORB that efficiently handles your data types.

Number of Processors — Number of Threads

As the cost of a processor continues to come down and as the architecture of the servers and the operating system allows, the number of processors in a typical C2 system continues to increase. Often 20 or more, and even up to 256 processors can be found in today's servers. Parallel processing of the ORB transactions greatly reduces the queuing affect at each stage of the processing; therefore, the time required for the transaction is also a function of the number of processors. The number of processors can sometimes exceed the number of threads.

So what? Look for a CORBA ORB that works well with a context-switching operating system.

Modern operating systems permit multiple threads of control within the same session. Like the number of processors, the number of threads can change the queuing affects at each stage of the transaction; therefore, the time to perform the transaction is also a function of the number of threads [10]. The number of threads often exceeds the number of processors.

So what? Look for a multi-thread capable ORB that can efficiently manage a thread pool (including protection against priority inversion).

Backplane Speed

The discussion on backplane speed is subtly different than bandwidth. Multiprocessor-capable servers use a high-speed (to extremely high-speed, i.e. 6 GHz) bus, generally referred to as a backplane, to facilitate communication between the processors. Like bandwidth, the speed of the backplane affects the time to transfer data between processors. When CORBA is used to communicate between processors within the same server, the time delays between processing stages are a function of the speed of the backplane. A server with a high backplane speed can communicate with much larger packets while not requiring an IP-type protocol (such as CORBA's Internet Inter-ORB Protocol), and therefore is far more efficient than a communications link with the same rate. In these cases, the efficiency of the ORB Core becomes the limiting factor.

So what? Look for a CORBA with an efficient broker, and one that is designed to adapt the protocol to the mode of communication.

Number of Servers — Normal Hashing or Perfect Hashing?

The more typical CORBA transaction is between two machines, a client and a server. Today's C2 systems often have several servers optimized for different operations: a communications server, a mission server, a database server, and a presentation server. The architecture may also include a combination of thin clients and thick clients. These architectures are referred to as N-tiered architectures.

In N-tier architectures, it is quite often found that the transaction process is further distributed, such that the ORB transaction can occur over three or more machines, for example: client, presentation server, and data server. The time delays occurring during the transaction are then a function of the number of servers the transaction is spread across. A higher number of servers complicates matters for the broker. Logical address resolution can become more difficult.

The broker uses a process known as normal hashing to dynamically calculate the route to the server. In a statically configured architecture, the designer can choose to short-cut this process by using a technique referred to as perfect hashing [4]. This process uses pre-calculated lookup tables to determine the route. Be cautioned though, this technique eliminates one of the principal advantages of CORBA: the flexibility to dynamically choose which of several servers will perform the method invocation as a way of doing load balancing. If the reduced flexibility is acceptable, be aware that the use of perfect hashing also increases the maintainability cost by requiring recalculation of the hashing tables every time the architecture is changed.

So what? If you have a fixed number of servers with fixed addressing, look for an ORB that allows perfect hashing.

Questions for the CORBA ORB Salesman

Armed with this new information about the factors to consider when selecting a CORBA ORB, you are ready to confront the ORB salespeople. Here are some sample questions you could ask:

- When I tested your competitor's product using a 7216 Byte packet size, we recorded an average one-way transaction time of 9.3241msec. Can you beat that?
- Do I have to use normal hashing or can I use perfect hashing?
- My entire system will be coded in Ada 95. What speed improvement can I expect using static invocation instead of dynamic invocation?

Here are some responses that would indicate you have a sales representative who knows the product and understands the implications of the alternatives:

- Was this a simple data structure or a complex one? What speed processor were you using? How many processors? How far apart were they? What was the speed of the backplane?
- Yes we support perfect hashing, but do you realize that you may get a minimal performance gain at the cost of less flexibility and more maintenance?
- The improvement will only be significant if you are using large-grained complex record types, and you run the risk of future software failure given an operating system upgrade on portions of the architecture.

Conclusions

Performance and flexibility are old rivals in computer architecture. Usually, design decisions made to achieve flexibility are detrimental to performance and vice versa. All of the decisions facing the architect therefore come down to how to best balance the needs of both goals.

Philosophically, it can easily be seen that any length of chain has a weakest link. Likewise, there is always a bottleneck in any process. One of the key roles of the software system architect is to understand, be able to detect, and manipulate the location of the bottleneck. Manipulating the location of the bottleneck is relatively straightforward: adding or taking away processors here or bandwidth there. In the case of CORBA for C2 systems, the architect must do the following:

- Understand the architectural and performance needs of the system.
- Understand the dimensions of variability within that envelope to select the critical factors for comparison purposes.
- Apply the above to the selection of a CORBA ORB implementation that can be effectively optimized for that architecture.

What quickly became evident during this research was that in the age of gigaflop computers, the time it takes to send a message across a LAN, approximately 1 msec, is considered an eternity. Since these communications are controlled by, facilitated by, and in most cases are conducted on behalf of the middleware, its performance becomes paramount.◆

References

1. Croak, T.J. "Application of Capacity Planning Techniques as Architectural Design Decision Aids for 3-Tier and N-Tier Software Architectures." DCS Dissertation. Colorado Technical University, 2000.
2. Seetharaman, K. "The CORBA Connection." Communications of the ACM 41.10 1998.
3. Schmidt, D.C. "Evaluating Architecture for Multithreaded Object Request Brokers." Communications of the ACM 41, (Oct. 1998): 54-60.
4. Schmidt, D.C. "Principles and Patterns of High-Performance and Real-Time Distributed Object Computing." ACM Symposium on Principles of Distributed Computing, 1997.
5. Gokhale, A. and D.C. Schmidt, "Measuring the Performance of Communication Middleware on High-Speed Networks." ACM SIGCOMM Computer Communication Review 26.4 (1996): 306-317.
6. Gokhale, A.S., and D.C. Schmidt. "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks." IEEE Transactions on Computers 47.4 1998: 391-413.
7. Tanenbaum, A.S. Distributed Operating Systems. Englewood Cliffs, N.J: Prentice Hall xvii, 614, 1995.
8. Kleinrock, L., "The Latency/Band width Tradeoff in Gigabit Networks." IEEE Communications 30.4, 4 Apr. 1992: 36-40.
9. Gokhale, A., and D.C. Schmidt. "The Performance of the CORBA Dynamic Invocation Interface and the Dynamic Skeleton Interface Over High-Speed ATM Networks." IEEE GLOBECOM '96. London, 1996.
10. Schmidt, D.C., et al. "A High-Performance Endsystem Architecture for Real-Time CORBA." IEEE Communications 14.2, (1997): 72-77.

About the Author

Thomas J. Croak is currently chief scientist of the Joint Missile and Air Defense unit of Computer Sciences Corporation. Dr. Croak is a principal investigator analyzing compliance of Battle Management Command, Control and Communications Systems with Department of Defense standards for architecture, services infrastructure, and design philosophy. Dr. Croak is retired from the Air Force, and among his positions he was the senior software engineer of the Air Force and managed the Air Force Software Technology for Adaptable and Reliable Systems (STARS) program for DARPA. Croak earned his doctorate in computer science from Colorado Technical University.

Computer Sciences Corporation
1250 Academy Park Loop, Suite 240
Colorado Springs, CO 80910
Phone: (719) 572-2588
E-mail: tcroak@csc.com

Over 40,000 Informed and Educated Readers Every Month
CrossTalk
Subscribe NOW!
www.stsc.hill.af.mil/crosstalk

Coming Events

January 27-31, 2002
2002 Western MultiConference
San Antonio, TX
www.scs.org

February 4-6, 2002
International Conference on COTS-Based Software Systems (ICCBSS)
Orlando, FL
www.iccbss.org

February 11-15, 2002
Application of Software Measurement (ASM 2002)
Anaheim, CA
www.sqe.com/asm

February 25-27, 2002
15th Conference on Software Engineering Education and Training (CSEE & T)
Covington, KY
www.site.uottawa.ca.cseet2002

March 19-21, 2002
Federal Office Systems Exposition 2002
Washington D.C.
www.fose.com

April 28 – May 2, 2002
Software Technology Conference 2002 "Forging the Future of Defense Through Technology"
Salt Lake City, UT
www.stc-online.org

May 13-17, 2002
Software Testing Analysis and Review (STAREAST 2002)
Orlando, FL
www.sqe.com/stareast

June 4-7, 2002
8th IEEE International Symposium Software Metrics (Metrics 2002)
Ottawa, Ontario, Canada
www.software-metrics.org