Web services architect, Part 3: Is Web services the reincarnation of CORBA?

Dan Gisolfi (gisolfi@us.ibm.com)
Solutions Architect, IBM jStart Emerging Technologies
July 2001

Contents:
A brief history of Communication Protocol Models
What is CORBA?
What is DCOM?
Basic RPC architecture
Subtle differences impact interoperability
Why CORBA and DCOM success is limited
What is needed for an improved RPC solution
Reliance on open Internet standards
The road to success
Recap
Resources
About the author
Rate this article

Related content:
Web services architect, Part 1
Web services architect, Part 2
More dW Web services resources

Even during these early stages of the evangelism of Web services, customers have already begun to ask how this technology differs from CORBA. Isn’t it just another form of distributed computing? In this installment of the Web services architect, Dan Gisolfi offers a brief overview of the differences between SOAP, DCOM, and CORBA and suggests a value proposition for Web services within the distributed computing realm.

Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.

In previous articles, I have discussed the vision of Dynamic e-business and the available technologies that enable us to achieve that vision, namely SOAP, WSDL, and UDDI. For the inquisitive reader the entire topic of Dynamic e-business is nothing more than another form of distributive computing. The justification for this new distributed computing model is cross-platform and cross-programming language interoperability. For the first time since distributive computing has been a mainstream concept, we have a solution based on open standards that can truly support interoperability.

Throughout my travels promoting Web services, a healthy skepticism always seems to arise among the people I talk to. Regardless of the audience, skepticism about Web services usually comes in the form of one of the following (and related) questions:

- How is Web Service Technology different from CORBA?
- Why will Web services succeed where CORBA failed?

I welcome these questions for two reasons. First and foremost, they reflect a familiarity with distributed computing that helps to frame the conversation. Distributed computing programming models have been mainstream in the IT industry for some time. I would guess that they have been around for the last two decades at the very least. As a result, the conversation can be focused on the shortcomings of previous solutions and future requirements for successful solutions.

Second, since we are not dealing with a new concept the associated risk for adopting Web services is low, which reflects its foundation on open standards and the pervasiveness of these standards in vendor solutions. The low risk is also a reflection of the real efforts that have been going on to advance interoperability for Web services (as opposed to the elusive promise of vendor interoperability that previous solutions have offered -- I'll have more on this later).

In the past, the Object Management Group (OMG) and its 700+ member companies have attempted to define how vendors needed to design object request brokers (ORBs) so that interoperability might be achieved. However, the reality has been that vendors compete on ORB implementations and thus there no motivation existed from a business perspective to achieve interoperability. In fact, the real ORB vendor business objective was to sell their solutions at both ends of the distributive computing application: *requestor* and *provider* nodes.

SOAP is a great distributive computing solution because it achieves interoperability through open standards at the specification level as well as the implementation level. But I am jumping ahead of myself. Lets step back and take a look at CORBA, DCOM, and, finally, SOAP. I will provide some key differences and present a case for why Web services based on SOAP technology encompass a better distributed-computing solution.

A brief history of Communication Protocol Models

Distributed applications require a protocol which defines the communication mechanism between two concurrent processes. Two communication protocol models for building such applications exist: *messaging passing/queuing* and *request/response*. While both messaging and request/response models have their advantages, either one can be implemented in terms of the other. For example, messaging systems can be built using lower-level request/response protocols. This was the case with Microsoft's Distributed Computing Environment (DCE) . For Remote Procedure Call (RPC) applications, the synchronous request/response design style is usually a natural fit.

In the 1980s, communication protocol models focused on the network layer, such as the Network File System (NFS) developed originally by Sun Microsystems -- which most networked Unix systems use as their distributed file system -- and Microsoft's DCE RPC application on Windows NT. In the 1990s the object-oriented programming community pushed for an Object RPC (ORPC) protocol which would link application objects to network protocols. The primary difference between ORPC and the RPC protocols that preceded them was that ORPC codified the mapping of a communication endpoint to a language-level object (see *A Young Person's Guide to The Simple Object Access Protocol* in Resources).

This mapping allowed server-side middleware to locate and instantiate a target object in the server process. Techniques such as mapping indices into an array or associating symbolic names as hash table keys were used to achieve the endpoint to object mapping. Prior to the advent of SOAP and Web services, Microsoft DCOM and CORBA's Internet Inter-ORB Protocol (IIOP) flavor of the General Inter-ORB Protocol (GIOP) were the dominating ORPC protocols in the industry.

What is CORBA?

The Common Object Request Broker Architecture (CORBA) is the Object Management Group's specification for achieving interoperability between distributed computing nodes. Their objective was to define an architecture that would allow heterogeneous environments to communicate at the object level regardless of who designed the two endpoints of the distributive application.

CORBA 1.1 was introduced in 1991 by Object Management Group (OMG). It defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). The ORB is the middleware that establishes the requestor-provider relationships between distributed objects.

CORBA 2.0 , adopted in December of 1994, addressed interoperability by specifying how ORBs from different vendors can interoperate (see Resources).

An ORB would receive an invocation message to invoke a specific method for a registered object. The ORB would then intercept the message and be responsible for finding an object that could implement the request, pass it the parameters, invoke its method, and return the results. In theory, the requesting node does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface.

A CORBA object is represented to the outside world by an interface with a set of methods. A particular instance of an object is identified by an object reference. The client of a CORBA object acquires its object reference and uses it as a handle to make method calls, as if the object were located in the client's address space. The 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 reply (if any) back to the client.

What is DCOM?

DCOM is the distributed extension of Microsoft's COM (Component Object Model) (see COM95 in

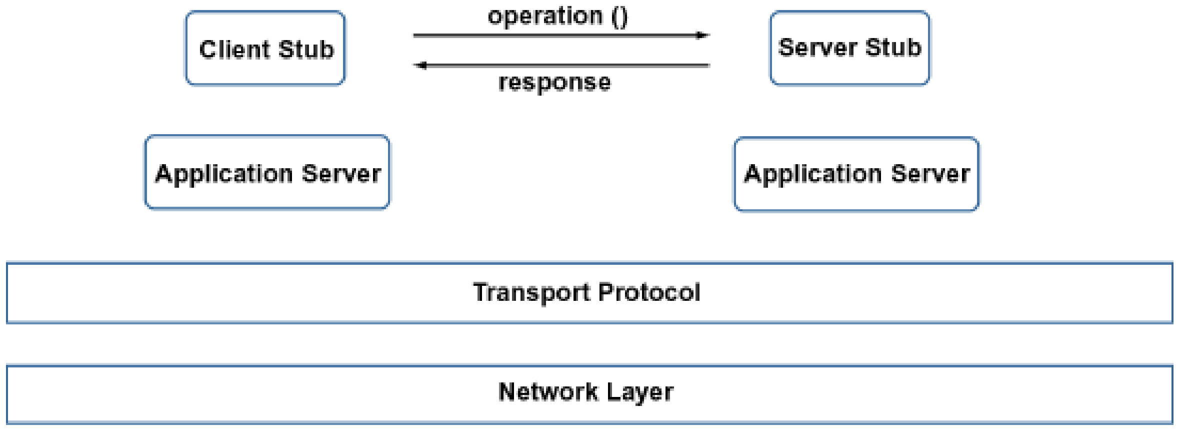
Resources), which builds an object remote procedure call (ORPC) layer on top of DCE RPC (see DCE95 in Resources) to support remote objects. A COM server can create object instances of multiple object classes. A COM object can support multiple interfaces, each representing a different view or behavior of the object. An interface consists of a set of functionally related methods. A COM client interacts with a COM object by acquiring a pointer to one of the object's interfaces and invoking methods through that pointer, as if the object resides in the client's address space. COM specifies that any interface must follow a standard memory layout, which is the same as the C++ virtual function table (see Rogerson96 in Resources). Since the specification is at the binary level, it allows integration of binary components possibly written in different programming languages such as C++, Java, and Visual Basic (see Resources).

Basic RPC architecture
In both DCOM and CORBA, the interactions between a client process and an object server are implemented as object-oriented RPC-style communications. Figure 1 shows a typical RPC structure. To invoke a remote function, the client makes a call to the client stub. The stub packs the call parameters into a request message, and invokes a transport protocol to ship the message to the server. At the server side, the transport protocol delivers the message to the server stub, which then unpacks the request message and calls the actual function on the object. In DCOM, the client stub is referred to as the *proxy* and the server stub is referred to as the *stub*. In contrast, the client stub in CORBA is called the *stub* and the server stub is called the *skeleton*. Sometimes, the term *proxy* is also used to refer to a running instance of the stub in CORBA. With respect to SOAP and Web services, we refer to the client stub as the *service proxy* and the server stub as the *service implementation template*. Table 1 illustrates these different names for the concepts.

Table 1: Client and Server components in different RPC architectures

RPC architectures	**Client Stub**	**Server Stub**
CORBA	Stub	Skeleton
DCOM	Proxy	Stub
Web services	Service Proxy	Service Implementation Template

Figure 1: Basic RPC Architecture



Subtle differences impact interoperability
DCOM and CORBA IIOP have many similarities. Both protocols use endpoint identifiers to identify a target object within the server-side middleware and they both use method identifiers to determine the signature of the method to be invocated.

Table 2: Implementation Attribute names in CORBA vs. DCOM

Implementation Attribute	**DCOM**	**CORBA**
Endpoint Naming	OBJREF	IOR
Interfaces / Object	Multiple	Single
Payload Parameter Value Format	DR	CDR

However, with these similarities come some differences that impact interoperability. As illustrated in Table 2, the three major differences are:

- **Naming of communication endpoints**: In ORPC protocols, some message representation of an ORPC endpoint is needed to communicate object references across the network. In CORBA/IIOP, this representation is called an Interoperable Object Reference (IOR). IORs contain addressing information in a portable format which any CORBA product can resolve to an object endpoint. In DCOM, this representation is called an OBJREF, which combines distributed reference counting with endpoint/object identification. Unfortunately, IORs do not correlate to OBJREFs, which results in an interoperability problem between CORBA and DCOM applications.
- **Support for multiple interfaces per object**: In CORBA, the interface identifier is implicit because only one object interface is supported, whereas DCOM supports multiple interfaces per object.
- **Format for payload parameter values**: In DCOM, the payload is written in a format known as Network Data Representation (DR). In IIOP/GIOP, the payload is written using Common Data Representation (CDR) format. Both DR and CDR deal with the differing data representations used on various platforms. It should be noted that some minor differences exist between these two formats which make them incompatible with one another.

Why CORBA and DCOM success is limited
Although CORBA and DCOM have been implemented on various platforms, the reality is that any solution built on these protocols will be dependent on a single vendor's implementation. Thus, if one were to develop a DCOM application, all participating nodes in the distributed application would have to be running a flavor of Windows. In the case of CORBA, every node in the application environment would need to run the same ORB product. Now there are cases where CORBA ORBs from different vendors do interoperate. However, that interoperability does not extend into higher-level services such as security and transaction management. Furthermore, any vendor specific optimizations would be lost in this situation.

Both these protocols depend on a closely administered environment. The odds of two random computers being able to successfully make DCOM or IIOP calls out of the box are fairly low (see Resources). In addition, programmers must deal with protocol unique message format rules for data alignment and data types. DCOM and CORBA are both reasonable protocols for server-to-server communications. However, both they have severe weaknesses for client-to-server communications, especially when the client machines are scattered across the Internet.

What is needed for an improved RPC solution
While CORBA and DCOM have their limitations, a distributed computing model is badly needed, and so they are widely used. In fact, with the advent of the Internet, businesses have a stronger desire to integrate with distributed applications outside of their enterprise. So what attributes are required for a successful distributed computing model? Well, for starters the solution must be vendor, platform, and language agnostic. Furthermore, it must offer more than the promise of interoperability; it must make strides at proving interoperability. Additionally, it must be simple for programmers to use the protocol and deploy applications. This requires easy access to client and server side implementations of the protocol. Simply stated, we need a new distributed computing model based on open Internet standards.

Reliance on open Internet standards
Enter the Web Services Technology stack, a set of open specifications that are either existing Internet standards or specifications that are widely accepted and are proceeding through the normal procedures to become standards. The basic stack is comprised of HTTP, XML, SOAP, WSDL, UDDI, and WSFL.

At the base of the stack we have HTTP, an RPC-like protocol that is simple, widely deployed, and firewall-friendly. Next, we have a common data representation language in XML, which is also extremely pervasive. SOAP, an XML-based messaging protocol is platform and language agnostic. It supports both message passing and request/response communication models. Like CORBA and DCOM, it requires an IDL. What it uses, WSDL, is an XML-based service IDL that defines the service interface as well as its implementation characteristics.

Let's revisit Table 2 and concentrate on the value HTTP and XML bring to Web services as a new distributed computing model. In HTTP, both the request and response messages can contain arbitrary payload information. HTTP headers are just plain text, which makes it easy to use by the average Internet programmer. Typically these headers contain a content-length and content-type. Furthermore, HTTP employs TCP/IP as the network communications protocol for its request/response messages. An HTTP client connects to an HTTP server using TCP. After establishing the TCP connection, the client can send an HTTP request message to the server. The server then sends an HTTP response message back to the client after processing the request. Simply put, HTTP is an elegant, payload agnostic transport that offers most of the connection management features found in CORBA and DCOM. It also makes use of URLs for object references that coincides with IORs and OBJREFs, found in CORBA and DCOM, respectively.

Since HTTP is payload agnostic, it does lack a mechanism for representing parameter values in the RPC messages. This is where XML comes into the picture. XML is a platform-neutral tagged-data representation language. It allows data to be serialized into a message format that is easily decoded on any platform. However, unlike DR and CDR, XML is simple to use, offers a flexible easy-to-extend data format, and is supported on virtually every computing platform. Once again, it is open and widely adopted. Table 2 depicts how HTTP and XML address the interoperability issues that plague CORBA and DCOM.

The Web Services Technology stack offers SOAP as the open standard ORPC for mapping application objects to network protocols (see Table 3). Although SOAP is not tied to a specific transport protocol, HTTP has become the early favorite among SOAP adopters. Using HTTP, a SOAP envelope uses XML as an encoding scheme for request and response parameters. A SOAP message is basically an HTTP request and response that complies with the SOAP encoding rules. A SOAP endpoint is just an HTTP-based URL that identifies a target for method invocation. Like CORBA, SOAP does not require that a specific object be tied to a given endpoint. Rather, it is up to the implementer to decide how to map the object endpoint identifier onto a server-side object. The namespace URI that scopes the method name in SOAP is functionally equivalent to the interface ID that scopes a method name in DCOM or CORBA.

Table 3: Interoperability Attributes for Web Services

Implementation Attribute	**Web services**
Endpoint Naming	URL
Interfaces / Object	Multiple - WSDL
Payload Parameter Value Format	XML

The road to success
Reliance on open, widely adopted standards is just part of the solution. We also need to make sure that the solution offers a high degree of interoperability and that implementations of the protocol are easily accessible. This is where I feel Web services has a promising future. Vendors are coming together in favor of the standards that comprise the Web services stack. Maybe it is due to timing and economics, or maybe it is due to the Internet's impact on programming models. Regardless of the cause, the result is very satisfying. Unlike the single-vendor implementation requirements that resulted from DCOM and CORBA solutions, vendors agree that it is in everyone’s best interest to define a distributed computing model for which applications can achieve interoperability.

Recently, IBM hosted an Interoperability workshop for a large group of SOAP vendors. Companies like Microsoft and WebMethods willingly participated to insure that their SOAP solutions would interoperate with other vendor solutions. This is a 180-degree cultural turnaround from the ORB battles of the 1990s. And this workshop was just the start. Another workshop, hosted by a different vendor, is already being planned.

The story gets even better when we introduce the notion of open source implementations. Implementations for the core of the Web services stack, namely HTTP, XML, and SOAP, are freely available through the Apache open-source community. And free implementations of WSDL exist from Microsoft and IBM. So the average programmer can openly acquire the necessary tools to quickly develop a distributed application and furthermore, they will rest assured that deployment dependencies for these applications will be easily and cost-effectively resolved by application users.

Recap
Is Web services the reincarnation of CORBA? No, at least I do not view the Web services programming

model as a reappearance or revitalization of CORBA. Instead, I see it as a new open solution that addresses the same distributed computing issue that CORBA addressed with the further objective of improving on some of CORBA's shortcomings.

The technology of Web services offers a new programming model for building distributed applications using open Internet standards. This new distributed computing solution exploits the openness of specific Internet technologies to address many of the interoperability issues of CORBA and DCOM. Specifically, Web services

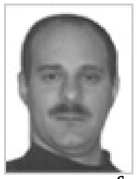
- uses HTTP to be firewall friendly and payload agnostic;
- employs XML as an encoding schema that is more widely adopted than DR and CDR;
- offers a free versus fee economic value proposition with regard to HTTP/SOAP server environments versus ORB frameworks;
- uses the pervasive Internet concept of URLs to address object identification; and
- offers more than a promise of interoperability. Vendors are actually working aggressively to prove that their SOAP implementations do interoperate.

Is Web services a better RPC? I think so, but only time will tell.

Resources

- Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.
- Read my first and second columns in this series.
- Read the Web Services Conceptual Architecture for an overview of Web services.
- Checkout real world adoption scenarios for dynamic e-business.
- Review the Simple Object Access Protocol.
- Read about the Web services Description Language.
- Visit the Object Management Group's site to learn more about CORBA..
- Learn how to integrate a CORBA ORB with WebSphere Application Server.
- WebSphere Application Server supports CORBA, Web services, and J2EE.
- Learn more about Universal Description, Discovery, and Integration by visiting its homepage.
- Look to see who is part of the XML Protocol Workgroup.
- More dW Web services resources.

About the author



A 13 year veteran of IBM, Dan Gisolfi holds a Masters Degree in Artificial Intelligence from Polytechnic University, and a BA in Computer Science from Manhanttanville College. Prior to 1999, his career was focused on software and product development ranging from Expert Systems, OS/2, and Secure Internet Payment Systems. As a member of the jStart (jump-Start) Emerging Technologies Team, he keeps his hands dirty in both the business and technical aspects of customer engagements. From Business Development Manger and Evangelist to Solution Architect and Contract Negotiator he gets to wear many hats. As jStart Team Lead for Web services, he is helping IBM drive the adoption of this emerging technology through real business solutions. You can reach him at gisolfi@us.ibm.com.

Discuss e-mail it!

What do you think of this article?

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

Comments? Send us your comments or click Discuss to share your comments with others.

Submit feedback