Like it or Not, Web Services *are* Distributed Objects!

Ken Birman
Dept. of Computer Science, Cornell University

Within the community developing the Web Services architecture and products, an increasingly schizophrenic message is emerging. Marketing materials assure us that Web Services are a breakthrough, offering unparalleled interoperability and comprehensive standards for associated technologies, such as transactions. They portray Web Services as a seamless interconnection layer that will propel computer-to-computer commerce to a previously inaccessible level. And they use language evocative of marketing for distributed object middleware.

Technologists are sending a somewhat different message. For example, in an essay entitled "Web Services are not distributed objects," Werner Vogels argues that Web Services will work well for important classes of applications, but he also cites significant limits. As Vogels sees it, the architecture is so centered on document exchange, and at its core is so simple, that many features taken for granted in object-oriented systems are fundamentally lacking. Examples include dynamic object creation and garbage collection, state management, dynamically created object references, and a variety of reliability and transactional mechanisms [Vogels03].

Both perspectives can't be correct.

It's easy to see how this situation arose. Web Services are the most recent in a long series of object oriented interoperability platforms, and mixes ideas from CORBA, J2EE and .NET, while exploiting XML and other Web-based document technologies. Developers using popular middleware platforms can transform a program object into a Web Services object, or access a remote WS object, at the touch of a button. Performance leaves something to be desired, but computers and networks have become astonishingly fast. Major application providers are planning to offer WS interfaces to their products. So it makes perfect sense that the marketing community would feel that finally, they've reached the promised land.

The technology community, in contrast, has an understandable emphasis on "facts on the ground" and the Vogels essay reflects the realities of an architecture focused at its core on using document exchange to access backend servers. This core has been extended with such mechanisms as RPC and asynchronous messaging, transactions (in several flavors), message queuing, a variety of roll-forward and rendezvous options, and event-based notification. But the primary usage case remains that of a client sending documents to a back-end service in a client-server or three-tier database architecture. The assumption is that the application can tolerate substantial delay before a response arrives, and mechanisms capable of introducing delays are scattered throughout the architecture. The more basic assumption is that it all boils down to moving documents around – whereas the most basic assumption of a distributed object system is that the world consists of programs and data: active and passive objects.

The gist of Vogel's essay is that even with all the contemplated extensions, Web Services are deeply mismatched with distributed object computing.

The dilemma underlying the debate is that the platforms one uses to create WS-compatible objects impose no such restrictions. There is nothing in J2EE or .NET that warns a user that an intended use of the architecture may be inappropriate. Indeed, much of the excitement reflects the realization that with Web Services, interoperability really *is* easier. Developers have long struggled with program-to-program interconnection and integration, and it is natural to applaud a widely adopted advance. Like it or not, Web Services are becoming a de-facto standard – for everything.

That's not all. Operators of web-based direct sales systems are turning to the WS architecture as a means of enlarging their markets. For example, Amazon.com has developed a web-access library whereby third-party application developers can access their datacenters from a diversity of end-user applications. An application could order thus supplies directly from Amazon.com, query the fulfillment system to track order status or billing data, etc. Both the vendor and the application developer benefit: Amazon.com enlarges its client base, while the developer avoids duplicating an enormous technology investment. Over time, Web Service components will play a critical role in tremendous numbers of end-user systems.

The challenge is to make such systems work reliably. Outages that plague human users of Web browsers don't cause much harm. With Web Services, outages could disrupt a computer-to-computer pathway buried deep within an application on which an enterprise has become dependent.

It is too easy to dismiss these concerns by arguing that the Web is extremely scalable and robust, but this ignores the way we use the Web. A human can deal with the many error conditions the Web exposes. Handling those conditions in a seamless, automated manner is an entirely different challenge. Moreover, when we take what was once a batch service or a Web site and transform it into a Web Service, there is no way to enforce appropriate patterns of use. What's to stop a Web client from trying to download Amazon.com's entire catalog? Today, the only answer is: "end to end, proprietary mechanisms."

Similarly, one might argue that none of these uses are what the architecture is intended to support. Not so many years ago, the major client-server architectures faltered over precisely this type of situation. Client-server technologies of the 1980's were widely seen as a kind of panacea, a silver bullet that would slay evil mainframe architectures. Enterprises fell over themselves in a kind of technology gold rush, only to discover that the technology had been oversold. Even today, the total cost of ownership for client-server systems remains excessively high: the number of system administrators remains roughly proportional to the size of the deployment.

Twenty years ago, a list like these comments might have seemed like an indictment of the technology, because we lacked solutions. Today, things are different. We know how to implement management tools and fault-tolerance mechanisms, how to replicate data and

functionality, and how to achieve high availability. We've had decades of experience with large-scale system monitoring and control, and are beginning to understand how to build solutions on an Internet scale. Peer-to-peer file sharing turns out to be illegal (and it doesn't work all that well, either), but spawned a new generation of technologies based on distributed hash tables and epidemic communication protocols. These offer remarkably stable, scalable tools for dealing with enormous numbers of components scattered over a network.

Not all the stories are positive. For example, the Web Services community decided not to adapt the CORBA fault-tolerance standard for their setting. This is a specification I know well; it was based on the virtual synchrony model colleagues of mine and I developed in work on the Isis Toolkit. The standard hasn't been a commercial success.

But the CORBA standard limits itself to lock-state replication of a deterministic server. Perhaps the issue is the way the technology was used, not the technology itself. Virtual synchrony, used in other ways, has been quite successful. Even today, Isis runs the New York Stock Exchange quote and trade reporting system (a role it has played since 1993), the Swiss Exchange, and the French Air Traffic Control system, and the US Naval AEGIS warship communication system, to name just a few. Leslie Lamport's Paxos protocol has been used to build file systems and scalable clusters. None of these examples uses lock-step replication of the type mandated by CORBA.

Every technology has its successes and failures. Used naively, any could fail. Used appropriately, these technologies could take the Web Services architecture to a new level. Moreover, doing so could greatly enlarge the Web Services market.

So what's the bottom line: Are Web Services distributed objects? *Of course they are!* The marketing people are listening to customers, and they want distributed objects. But Vogels is right, too: Web Services, as currently conceived, won't suffice.

It's time for the Web Services community to come to grips with the needs of their customer base. One can justify solutions that make 90% of the customers happy but leave 10% dissatisfied. Indeed, a solution that tries to do better will probably overreach. But you can't get there if you close your eyes to the way the customers are likely to use the technology. Will the Web Services community have the wisdom to tackle the tough issues before circumstances force it upon them?

Bio: Ken Birman, a Fellow of the ACM, received his Ph.D. from U.C. Berkeley in 1981, and has worked on reliability and scalability issues in distributed systems since starting his research career. He is the author of many articles on the subject. His book, *Reliable Distributed Systems: Technologies, Web Services, and Applications* will be published by Springer Verlag in Fall 2004.

[Vogels03] Werner Vogels. Web Services are not Distributed Objects. IEEE Internet Computing, Vol. 7, No. 6, pp 59-66, November/December 2003. Online at http://weblogs.cs.cornell.edu/AllThingsDistributed/archives/000343.html