Kenneth P. Birman

Like It or Not, Web Services Are Distributed Objects

Despite the push to adopt Web services as the universal OO architecture, the Web services reliability model ignores many real-world issues routinely encountered by users.

LISA HANEY

Within the community developing the Web services architecture and products, a disjointed message is emerging. It points to a serious problem for potential users, as well as for platform vendors. If they don't address this confusion now, we're likely to see a real clash between user expectations and platform characteristics, as Web services systems begin to roll out in the near future.

In language evocative of marketing campaigns for distributed object middleware, Web services marketing materials assure us that Web services offer unparalleled interoperability and comprehensive standards for associated technologies (such as transactions). Vendors routinely portray Web services as a seamless interconnection layer that will propel computer-to-computer commerce to a previously unattainable level of interoperability and integration.

Technology developers responsible for building the platforms are sending a different message. For example, in a Nov. 2003 Weblog essay "Web Services Are Not Distributed Objects," Werner Vogels, at the time a researcher in the Department of Computer Science at Cornell University, argued that Web services will work well for important classes of applications but also cited significant limitations. As he saw it, the architecture is so centered on document exchange and at its core adopts such a restrictive computing model that it's likely to be inappropriate for many of the uses software engineers might attempt when developing OO systems. Missing functionality includes life cycle services, dynamic object creation and garbage collection, state management, dynamically created object references, and a variety of reliability and transactional mechanisms. Vogels, who now directs systems research at Amazon.com, later published the essay in [1], attracting significant attention.

Both perspectives can't be correct. If Web services are the Holy Grail of interoperability, they can't also be so fundamentally restrictive as to be unusable for common OO programming tasks. After all, OO systems became popular precisely because they respond to the basic needs of developers confronting tough interoperability challenges.

It's easy to see how this situation arose. Web services are the most recent in a long series of OO interoperability platforms, mixing ideas from the Common Object Request Broker, Java 2 Platform, Enterprise Edition (J2EE), and .NET, while standardizing around XML and other popular Web-based document technologies. Developers using middleware platforms can transform a program object into a Web services object or access a remote Web services object at the touch of a button. Although the performance of these remote objects leaves something to be desired, computers and networks have become astonishingly fast, and most major application providers are planning to offer Web services interfaces to their products. Even the U.S. military is betting that "Enterprise Service" architectures could be the key to building future global information grids linking troops, planners,

intelligence databases, and sensors in the field. Meanwhile, manufacturers of a new wave of small wireless sensors are standardizing around Web services interfaces.

With such broad uptake already under way, it comes as no surprise that the technology marketing community views Web services as the promised land.

In contrast, the technology *development* community emphasizes facts on the ground, and Vogels' essay reflected the realities of an architecture focused on document exchange being used to access back-end servers. This core has been extended with such mechanisms as remote procedure calls, asynchronous messaging, transactions (in several flavors), message queuing, 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 pervading the architecture is that the application can tolerate substantial delay before a response arrives, and mechanisms capable of introducing delays are scattered throughout the architecture. An even more basic assumption is that Web services boil down to moving documents around—whereas the most basic assumption of a distributed object system is that the world consists of programs and data, or active and passive objects.

Vogels wrote that even with a variety of already contemplated extensions, Web services are certain to remain deeply mismatched with distributed object computing. Nobody doubts that Web services can be extended to address these uses, but it hasn't happened yet and isn't even on the horizon.

The dilemma underlying the debate over the true nature of Web services is that the platforms one uses to create Web services-compatible objects impose no such restrictions. There is nothing in J2EE or .NET warning a user that an intended use of the architecture may be inappropriate. Indeed, much of the excitement about the promise of the technology reflects the realization that with Web services, interoperability really is easier. One can connect literally anything to literally anything else, whether it means client software talking to a back-end server or a mobile Web application talking to an ad-hoc network of wireless sensors. 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 Web services architecture as a means of enlarging their markets. For example, Amazon.com has developed a Web-access library whereby third-party application developers access Amazon's datacenters through diverse end-user applications. An application could thus order supplies directly from Amazon.com and query Amazon's fulfillment system to track order status or billing data. The vendor and the application developer both benefit. Amazon.com increases its client base, while the developer avoids having to duplicate an enormous technology investment. Over time, Web services components will play a critical role in

Vogels wrote that even with all the contemplated extensions, Web services are certain to remain deeply mismatched with distributed object computing.

large numbers of end-user systems.

To a substantial degree, this is why Web services may become as ubiquitous as the Internet itself, penetrating into everything—the payroll system in your company, the order-fulfillment system at Amazon.com, the computers running your doctor's office and the local pharmacy, the on-board system in your car for finding the quickest route home from work, the software controlling the traffic lights, and NASA's next generation of Mars rovers.

The challenge for us as technology developers and leaders is to make these systems work reliably. Computer-to-computer outages of the sort that plague human users of Web browsers don't cause much trouble. But with Web services, outages could disrupt a computer-to-computer pathway buried deep within a critical application on which an enterprise depends in ways it may not even know about.

Lacking a good reliability (and availability) story, Web services platform developers seem to be suggesting that these uses aren't what the architecture is intended to support. Not many years ago, the major client-server architectures faltered over precisely the same type of situation. Client-server technologies in the 1980s were widely viewed as a silver bullet that would slay evil mainframe architectures. Enterprises fell over themselves in a 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; for example, the number of system administrators remains roughly proportional to the size of the deployment.

The curious thing is that we know how to solve these technical problems. We know how to implement management tools and fault-tolerance mechanisms, replicate data and functionality, and 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 capable of spanning the entire Internet. Peer-to-peer file sharing spawned a new generation of technologies based on distributed hash tables and epidemic communication protocols. They offer remarkably stable, scalable tools for dealing with enormous numbers of components scattered across a network. And we're quickly learning how to secure these technologies, so disrupting them would be far more difficult than was the case for previous generations of networked platforms.

Not all we've learned is positive, however; each technology involves successes and failures. Used naively, any of the mechanisms I've cited here could fail. Used appropriately, they could take the Web services architecture to a new level of stability, reliability, and trustworthiness. Moreover, doing so could greatly enlarge the Web services market.

Are Web services distributed objects? Without a doubt. The marketing people promoting Web services have been listening to their customers, and the customers need, want, and expect distributed objects. But Vogels is right, too; Web services, as conceived today, won't yield the trustworthy platform those customers need and expect. Today's Web services platforms can't support distributed objects.

It's time for Web services developers and vendors to come to grips with the needs of their customer base. One can justify solutions that make 90% of the customer base happy but leave 10% dissatisfied. Indeed, a solution that tries to do better would probably be overreaching. But you can't get there if you close your eyes to the way the great majority of customers are likely to use the technology. The Web services community must ask itself whether it has the wisdom to tackle the tough issues before a tsunami of dissatisfied users demands it. C

Reference

1. Vogels, W. Web services are not distributed objects. *IEEE Internet Comput. 7,* 6 (Nov.–Dec. 2003), 59–66; weblogs.cs.cornell.edu/AllThingsDistributed/archives/000343.html.

Kenneth P. Birman (ken@cs.cornell.edu) is a professor in the Department of Computer Science at Cornell University, Ithaca, NY.