Web services were born of the simple observation (apparently made independently by several people at about the same time) that XML could be used as a vehicle for issuing request and response messages across servers. From this humble beginning, they are on the way to transforming a huge portion of the distributed systems used for commerce and information services.

**Why Web Services?**

As defined by the World Wide Web Consortium, Web services are

- identified by URIs;
- accessible via standard Web protocols;
- capable of sending, receiving, and acting on XML-based messages; and
- capable of interacting with applications and programs that are not directly human-driven user interfaces.

To the most idealistic adherents, Web services are the ultimate unification of the Web, databases, XML, and distributed systems — the one true path for bringing electronic data interchange, transaction systems, and business-to-business services into the 21st century. To the most cynical detractors, however, Web services are just a slower, more complicated reinvention of the wheel of distributed infrastructure. The truth rests in a more pragmatic middle ground that is leading many to adopt Web service frameworks.

In large IT organizations, heterogeneous computing systems are a standard part of doing business. In these settings, the wealth of tools that exist — and continue to be developed — for creating Web service frameworks can greatly simplify software development. Indeed, such tools bring distributed services programming within the reach of just about any programmer. Many application designers and developers are drawn to Web services because they finally provide remote-invocation and document-passing standards that everyone can agree on. They also bring to an end the unfortunate era of screen scraping and other
ad hoc techniques for combining user- and infrastructure-oriented functionality. Done properly, Web service frameworks add value to existing infrastructure and applications by allowing them to be integrated at a level of abstraction that was previously impractical because of competing, divided standards and noninteroperable proprietary approaches.

Nonetheless, the nearly assured widespread acceptance of Web services brings its own set of challenges to the middleware and tools that must support it. Some of these are almost too obvious to state — coping with XML overhead (with respect to space, processing time, and bandwidth); reducing complexity while improving reliability in the sea of necessary software components; and of course, creating good tools that let developers generate messages, bindings, and so on without having to program in the underlying “assembly language” of raw XML.

While these are important topics, this issue of IEEE Internet Computing focuses instead on some of the broader architectural and design challenges Web services present for middleware — in both the short and medium term.

The Articles
To begin the theme section, Dalal et al. provide a technical overview of a recently completed cornerstone standard for Web services in “Coordinating Business Transactions on the Web.” Transaction coordination is a necessary ingredient in most commercial Web-based applications. This article illustrates the rapid transition from research and advanced development to practice in constructing transaction frameworks that make sense for Web services. Web transactions play the same role as classic transactions — coordinating interdependent updates across multiple data sources — but they must do so using distributed protocols that cope with failures, long-duration transaction times, and the uncertainties of ad hoc encounters among autonomous systems. Given all this, BTP is surprisingly straightforward and usable.

In “The Self-Serv Environment for Web Services Composition,” Benatallah et al. describe a prototype for a system that addresses the need for composing higher-level services and applications from sets of services provided by different enterprises. This is a forward-looking investigation into issues and problems that are at the beginning stages of standardization and commercialization. The research reported here aims to provide a more declarative and formalizable toolset to cope with the intrinsic complexity of composition on the Web.

In “Scalable Filtering of XML Data for Web Services,” Felber et al. attack some classic lower-level infrastructure issues while addressing some new twists. In the old days, routers and switches could automate the routing and delivery of packets with minimal overhead (via simple inspection of addresses and ports, for example) in determining the appropriate software service, but such routing and delivery is no longer adequate. SOAP and related XML-based schemes require further decoding to bind the many kinds of requests that a server (or set of servers) might support to the corresponding service components — potentially leading to serious bottlenecks. The authors introduce some clever algorithms that offer promising solutions to problems that could otherwise become serious impediments to Web service scalability.

Finally, in “Infrastructure for E-Government Web Services,” Medjahed et al. describe the design for a distributed social services system. Web services can potentially simplify the building of distributed applications, and the authors show an impressive and useful system that realizes this promise. At the same time, this case study reveals the types of standards and middleware components that are still needed before developing such systems becomes as routine as Web services advocates have predicted.

In Future Issues
This issue kicks off a new feature in IC: In addition to the peer-reviewed articles that appear in this theme section, we will bring you a related article in each issue in 2003 as part of the magazine’s new Web services track. Future articles will explore topics such as lower-level toolkits for XML processing and architectures for virtual enterprises. We welcome additional submissions to the track. See www.computer.org/internet/dept.html for details.

Doug Lea is a professor of computer science at the State University of New York, Oswego. He writes about and develops infrastructure for distributed and concurrent object systems. Contact him at dl@cs.oswego.edu.

Steve Vinoski is vice president of platform technologies and chief architect for IONA Technologies. He focuses on building real-world standards-based distributed middleware systems and infrastructure. Contact him at vinoski@ieee.org.