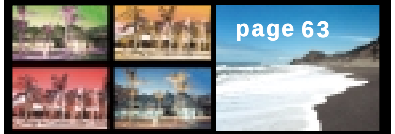
WEB SERVICES: THE NEW KILLER APP page 66

October 2001 Volume1 Issue 1

WebServices JOURNAL

PREMIER ISSUE

WSJ2.COM



page 63

OCTOBER 24–25, 2001

web services conference & expo EDGE

SANTA CLARA CONVENTION CENTER, SANTA CLARA, CA

By Jonathan Cortez page 46

BUILDING WEB SERVICES USING Microsoft .NET

How to Harness the Connectivity of the Internet

SYS-CON MEDIA

BEA

www.bea.com/download

BEA

www.bea.com/download

IONA

www.iona.com

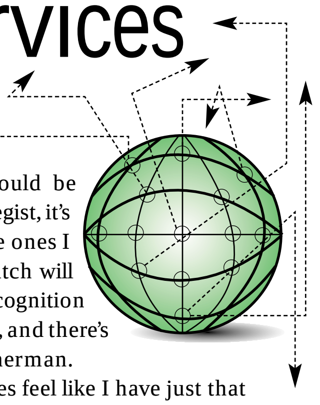
EDITORIAL OFFICES
SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9637
SUBSCRIBE@SYS-CON.COM
WEB SERVICES JOURNAL (ISSN# 1535-6906 IS PUBLISHED MONTHLY (12 TIMES A YEAR) BY SYS-CON PUBLICATIONS, INC., 135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
PERIODICALS POSTAGE PENDING
POSTMASTER: SEND ADDRESS CHANGES TO:
WEB SERVICES JOURNAL, SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645

A World of Web Services

written by
Sean Rhody

Author Bio:
Sean Rhody is the editor-in-chief of Web Services Journal. He is a respected industry expert and a consultant with a leading Internet service company.

SEAN@SYS-CON.COM



I sometimes think that the best job in the world would be strategic thinking. Every time I see a quote from a strategist, it's always five years out and full of promise. You know the ones I mean. It goes something like this: "By 2005 every wristwatch will have a Pentium 7 processor and 2 GB RAM, with voice recognition and holographic screen projection." It seems like nice work, and there's no burden of delivery. It's even better than being the weatherman.

At a time when Web services is in its infancy, I sometimes feel like I have just that job. One of the interesting facets of editorial work is setting an editorial calendar and selecting topics. Doing that for this first year of ***Web Services Journal*** has been akin to looking into my crystal ball and peering into the future.

Some things are easy to see – we're looking at a world of Web services deployed with technologies such as SOAP, UDDI, and XML. We'll be talking a great deal about the overall architecture of delivery, focusing on .NET and SunONE, among others.

Other things, though, are a little less clear. For Web services to become widespread, we need compelling business reasons that will drive adoption. There are several possibilities that loom on the horizon. Even though dot-com fever has cooled, the Internet still exists, still functions, and still connects millions of people and thousands of companies. The Internet has irreversibly altered the way companies do business, and progress continues unabated.

As businesses become increasingly dependent upon one another, they will continue to make efforts to tighten their supply chains. They may accomplish this with Web services, linking their business systems and providing real time responsiveness to the smallest of issues.

Peer-to-peer solutions may also drive the adoption of Web services. In a sense, Napster was one of the first peer-to-peer Web services – and it may be a consumer rather than a business focus that truly launches Web services.

Business-to-Consumer is an additional avenue for Web services. The discovery capability of the technology may finally make it possible for intelligent agents to break into the mainstream. Finally, we may be able to have our computers go out on the Web, find vendors who provide the services we're interested in, determine the lowest bidder (or other criteria, such as immediate availability), and conclude a transaction.

It may also be the need to interact with other companies at a process level rather than at a transactional level. Business conditions change frequently – too frequently in some cases for transactions to be fully defined. We might see Web services ride the tide of business process management, providing the infrastructure, or even the structure itself, of collaborative business.

Underlying all of this will be a plethora of technologies. As is typical of the information technology world, there are competing standards and platforms. Which in a way is not all bad. The reality of Web services is that we are pursuing a neutral course in terms of data and data structure, and layering services on top of transformable, machine-independent data. The ability to select a particular platform to provide Web services is an advantage to all – it future-proofs the existing and often enormous investment companies have made in their operational platforms.

Truly, we don't know which direction Web services will come from. In all likelihood it will be a combination of all of those avenues driving the need for standards and processes. It's impossible to say which standard or standards will prevail – Web services exists today, but not nearly in the abundance that they will in the future. But hey, since this is a strategy piece, I'll make my predictions.

By 2005 we'll see a world where personal agents interact with Web services to obtain content, products, and services. Businesses will build competitive advantages using Web services to improve their supply chain management, and will use business process management services, brokered by Web services, to adapt their business to constantly changing conditions.

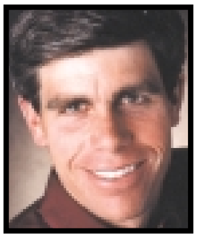
Like I said, the good thing about being a visionary is the lack of accountability. By 2005, no one will remember my prediction and we'll be concentrating on a world where Web services are endemic. And then I can make my 2010 predictions. Stay tuned for a world of Web services.

Web Services From the Inside Out

The buzz around Web services is enormous. Every day more and more developers are asking themselves if Web services should be used in their next project. Microsoft views the Internet and Web services as a complete development platform. Other companies view it as a set of complementary protocols. ***WSJ*** asked seven leading *i*-technology players for their expert opinions on the world of Web services…

WSJ asked those at the very heart of the emerging Web services paradigm: What are Web services? How might they change business as we know it?

Andy Roberts, Chief Technology Officer
Bowstreet, Inc.
Portsmouth, New Hampshire



"Web services are software comp-onents exposed on the Web using standard protocols such as HTTP and data formats such as XML. Web services enable software systems to call each other over the Internet, as well as on intranets, independent of the 'back-end technology' that's being used.

The generic markup concept behind XML is about separating information from its descriptive metadata, so that people can repurpose information in different contexts, using different devices. Web services is about breaking applications into pieces, so that units of functionality can be repurposed in similar manner.

By exposing functionality using standard XML data formats, rather than hiding behind a particular fixed presentation for humans to view via a browser, that functionality can be repurposed for different uses by different applications on different platforms in different contexts. Web services thus enable business processes from different providers to be combined in new and different ways. For example, a product manufacturer can provide product configuration, shipping, and financing services, all rolled up into one service, even though they are provided by different suppliers.

Without Web services, the Web enables individual businesses to reach out to customers, but only through 'silos' or 'one-to-one' connections. Web services enable a new kind of virtual business, brought about by connecting multiple business processes together within a software node, and redeploying it as composite, or aggregated, service."

Andy Roberts

Web services enable a new kind of virtual business to exist that connects multiple business processes together within a software node, and redeploy it as composite, or aggregated service.

David Litwack, President & CEO
Silverstream Software, Inc.
Billerica, Massachusetts

"Web services is an important technology. However, the really imp-ortant thing that's hap-pening is that the industry is going through a once-in-a-decade transformation to a new application para-digm – a new way of building applications.

David Litwack

What's happening today is we have a set of very important technologies – Java, J2EE, XML, SOAP, WSDL, HTTP, the Internet – that have culminated to a level of maturity where we can think about building the applications in a completely different way….Web services is certainly a key piece of that.

I'd like to draw a comparison with client/server. Client/server wasn't a specific technology. It was the culmination of a set of technologies — graphic user interfaces, relational databases, networks, and personal computers — that resulted in a new way of building applications. And that way of building applications was very important because it was a clean separation of information from the user interface. It transformed the way we use computers in businesses from what had been predominantly an administrative or clerical function, to having the corporate repository of information available to everybody that had a computer on their desktop. That was the revolution.

What's happening today is we have a set of very important technologies — Java, J2EE, XML, SOAP, WSDL, UDDI, HTTP, the Internet — that have reached a level of maturity where we can think about building the applications in a completely different way. The Gartner Group calls this a services-oriented architecture and Web services is certainly a key piece of that. The important thing about a services-oriented architecture is a clean separation of the transaction, information, or service from the audience where it's supposed to be delivered. Put another way, it turns the traditional design paradigm backward, the traditional being: architecting in who the audience for the application will be. In a services-oriented architecture you shouldn't know or have any design criteria that involves knowledge of who the audience is. That is so important today, because through the Internet we can deliver

IBM

www.ibm.com/websphere

applications to people we've never met or to devices that haven't been invented yet. We can deliver applications inside or outside a firewall.

So a services-oriented application meets the needs of e-businesses in the Internet age because it allows us to build applications that can be flexibly deployed in the future to unknown future audiences.

Web services are great but we don't want to deliver Web services in the same way to everybody. We want to tailor Web services depending on who the users are – what their jobs are, whether they're high net worth individuals, or retirees, whether they're business partners, corporate customers, or your own salespeople, whether they're inside or outside the firewall. We may increasingly tailor Web services whether it's the weekend or the week, or based on what device they are connected from. Someday, with GPS in wireless devices we may want to tailor the service depending on where they are, because with GPS devices we'll know where they are within 10 meters. These are all parts of the services-oriented puzzle."

David Clarke, President & CEO
Cape Clear Software Inc.
Walnut Creek, California

"For us it's all about opening up development, with a small d, to a much wider community of developers than those who only understand Java and Enterprise Java. The application server vendors still think of Enterprise Java as the central and fundamental programming model – so although they're beginning to provide SOAP stacks, they tend to view SOAP as just another way of connecting into the app server. 'We support TCP, IOP, RMI, and SOAP,' they're saying – so they seem to view SOAP as a protocol instead of, as we do, as a world-shattering new paradigm."

David Clarke

The application server vendors still think of Enterprise Java as the central and fundamental programming model...they seem to view SOAP as a protocol instead of, as we do, as a world-shattering new paradigm.

Simon Phipps, Chief Technology Evangelist
Sun Microsystems, Inc.
Santa Clara, California

"Web services is a new name for an old idea. It was called RPC, it's been called IOP, it's been called COM, it has been called all sorts of things. This time it's going to be big because it's going to allow linear cost growth while providing access to exponential business opportunity.

Companies are going to be able to discover each other over the Web and automatically initiate trading behavior. We're right on the cusp of that at the moment. The place beyond that is what happens when Web services meets wireless technologies, at which point the consumer will come back into the picture again, and we'll begin to see consumers being able to use the context of where they are to have options presented to them by the swarm of devices, software, and networks around them.

The place where we're headed to after that — this is probably about 3 years out — is a place of connected Web-smart services being conveyed over mixed media like over wireless, also over high speed connections into the home, and over 3G connections into more powerful wireless devices. The mixed nature of the client space in that world is what makes open standards even more important."

Simon Phipps

Companies are going to be able to discover each other over the Web and automatically initiate trading behavior. We're right on the cusp of that at the moment

Tyler Jewell, Principal Technology Evangelist
BEA Systems, Inc.
San Jose, California

"Developers find it incredibly easy to start working with BEA technology. It's not just about our application server and implementation of servlets, JSPs, and EJBs any more, it's about enabling developers to leverage their J2EE knowledge into a variety of different areas including Web services.

Our clients are definitely interested in Web

Web Services Edge 2001

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

WEB SERVICES CEO PANEL

LITWACK
CEO
SilverStream

O'CONNOR
President
Sonic Software

KUTAY
CEO
AltoWeb

O'TOOLE
Chairman
Cape Clear

SLAMA
CEO
Shinka Technologies

MORRIS
CEO
IONA Technologies

OCTOBER 22-25, 2001
Santa Clara Convention Center
Santa Clara, CA

SpiritSoft

www.spiritsoft.net/downloads

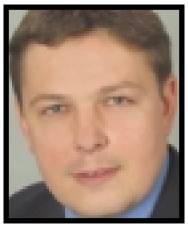
Tyler Jewell

BEA actually views Web services as an extension of J2EE – it's just an exposing of the existing applications and programs that they've already written, but in a new format.

services, but when we go and talk to BEA clients they're looking for ways to use Web services with J2EE. I know there's been a lot of talk about Web services as a new platform, but BEA actually views Web services as an extension of J2EE – it's just an exposing of the existing applications and programs that they've already written, but in a new format. One of the big things that we're doing with WebLogic 6.1 is finding ways to take programming expertise that Java developers already have and converting that directly to Web services expertise.

So with WLS 6.1 we now have point-and-click autogeneration from EJBs to Web services, and we have that available today."

Barry Morris, CEO
IONA Technologies PLC
Dublin, Ireland



"There is a big difference between service-oriented architectures in the abstract and Web service implementations in the real world. IONA has been building service-oriented systems for ten years and has a deep knowledge of the technology. Our systems have always been built on standards but we are embracing a new set of standards,

Barry Morris

We are seeing activity short term in Intranet implementation and expect real internet-based integration around Web services in the next six to twelve months.

such as SOAP, J2EE, and XML, and we are utilizing them in our Web services products.

IONA now has developed XMLBus, a Web services development system that is all about service-oriented architectures built in XML. We have taken a leadership role in the new standards bodies defining SOAP, UDDI, WSDL etc. We are also active at the higher level, at the business process choreography level, with support of ebXML and Rosettanet and that level of standard. XMLBus was created to build exactly those kinds of systems.

Our large customer base is moving increasingly toward services based on standards like XML and SOAP. We are seeing activity short term in intranet implementation and expect real Internet-based integration around Web services in the next six to twelve months. IONA is committed to meeting the needs of the large enterprise market. Toward that end we have recently introduced the IONA B2B Integrator, based on our acquisition of a leading firm in Business Process Collaboration. The combination of B2B Integrator, XMLBus, and our unique expertise in standards-oriented architectures, positions IONA for leadership in the Web services market."

Uche Ogbuji, CEO
Fourthought, Inc.
Boulder, Colorado

"Web services is not as glamorous as its usual billing has it. It is merely a way to assemble a toolkit that allows rapid development and deployment of collaborative applications. It takes advantage of Internet protocols for inexpensive implementation and XML data formats for extensible expression of data that is readily processed by people or agents. Both of these core technologies also bring the advantages of broad adoption and support in almost every vendor tool released in the past year or so.

Where the fruits of innovation are conveniently dispensed on the Web, Web services will lower the barrier to deploying such innovations. Service providers will be able to get their products in place rapidly, update them with relatively little pain, and advertise them to a potentially unlimited marketplace.

However, there are dangers. Many of the champions of Web services have every reason to combine the enticing openness of Web services with clever tricks to keep developers tied to their products. The best safeguard against this is to minimize the coupling of systems using Web services as much as possible. This means not insisting on quirks of data format, encoding or protocol, but allowing strong support for

Uche Ogbuji

Where the fruits of innovation are conveniently dispensed on the Web, Web services will lower the barrier to deploying such innovations.

negotiating all these factors. It also means leaning on other open standards as much as possible: for instance, service descriptions and classifications might better take advanatge of the well-established resource description framework (RDF) rather than reinvented XML formats.

Fourthought, being providers of software and solutions for XML applications, is very active in the community trying to avoid these pitfalls, and finding the right practical mix of Web services technologies for our clients. "

CONCLUSIONS

Distributed computing technologies have been around for a while and so, to that extent, the ideas behind Web services are nothing new.

But anyone looking at Web service technologies for some particular technical characteristic that's getting people excited about them won't find it. The reason that Web services are so exciting isn't any particular technical quality they have. Rather, it's the unprecedented agreement that the new paradigm is getting from so many parties who traditionally have been business rivals.

There are still disagreements, but a standard foundation is established and that foundation is growing every day.

One note of caution, amid all this optimism and excitement: there's still a great deal to be sorted out in the areas of trust, security, payment, and so on. It's precisely for this reason that the places where companies are gaining the greatest benefits from Web services at present are within the enterprise.

This will undoubtedly change. There's still a lot of work to do, but companies can and are using Web services today. And as for the consumer play...well – the future, as they say, lies ahead.

Sonic Software

www.sonicsoftware.com

For nearly 160 years, Dun & Bradstreet (D&B) has been collecting and providing information and analysis on public and private companies worldwide. As a business information company, D&B relies heavily upon technology to support data that produces up-to-the-minute decisioning products. In fact, the D&B global database is updated more than 1 million times a day and D&B owns and maintains one of the world's largest privately-run computer networks. In addition to collecting and aggregating information on more than 63 million companies worldwide, D&B offers products and services that help to analyze and predict a businesses' behavior.

D&B information is available in a number of formats, languages, and levels of detail that allow you to obtain the data that you need. For example, if you're going to be performing services for a company, you may be concerned with how quickly it pays its bills. You may also be concerned with the financial status of a company and want to know what its reserves are before you commit a significant amount of resources to working with it. Or, you may want to know who your largest suppliers are so that you can negotiate volume or corporate discounts. Likewise, you may want to know what your most profitable customer segment looks like and then target your marketing efforts to similarly profiled companies.

D&B can provide the answers to these questions in a standardized format that ranks individual companies, and also ranks them within their industries. D&B tracks this information in 209 countries.

When D&B was first incorporated, the company provided information via paper reports and later as faxed information. As the years progressed, D&B began to maintain and distribute its information via dedicated leased lines and private networks. The rise of the Internet, however, offered additional mechanisms for distributing the information. Information that 160 years ago took days or weeks to gather, can now be obtained via the Web from D&B's Internet site (www.dnb.com) by downloading the desired product or service.

Of course, visiting a Web site to download information is not really all that interesting in terms of Web services. What makes D&B interesting is that in addition to the Web site channel, many of the company's flagship products are now available via a Web service called the D&B Global Access Toolkit.

WSJ recently met with executives from D&B, to investigate how Web services are viewed by them and how well the D&B Global Access Toolkit fits our definition of Web services.

As companies begin to integrate their supply chains via Web services, the need for up-to-date information is crucial in the decision-making process.

"D&B has been a critical part of commerce for more than 160 years. Our customers needed a streamlined process for accessing D&B global data in real time. That's why we developed the D&B Global Access Toolkit" said Pat Winchester, Vice President of Global Products Management, D&B.

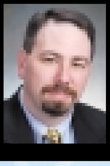
The D&B Global Access Toolkit provides the ability to easily integrate standardized, global data into existing applications and decision support systems while significantly reducing customers' development costs and time. Essentially a Web-based pipe to the D&B global database using XML standards, the Toolkit integrates D&B data with existing customer information, establishing D&B global data as core components of global or enterprise-wide decision making processes.

In today's "Internet time" economy, D&B information is required ASAP to verify a new business partner, make credit decisions, or evaluate a new supplier. Likewise, companies are working hard to reduce costs. "The D&B Global Access Toolkit allows us to offer our customers real-time access to D&B information, reducing development costs and business risks," noted Winchester, "providing our customers with better service is what Web services are about."

Since its release in May 2001, adoption of the Tookit has grown steadily. Although exact numbers were not available regarding adoption and usage, D&B is encouraged by current usage.

"The D&B Global Access Toolkit significantly expands D&B's online global data delivery capabilities, an important component of the company's aspiration to become a growth company with an important presence on the Web," said Cynthia Hamburger, the CTO of D&B.

WRITTEN BY
SEAN RHODY



Sean Rhody is the editor-in-chief of **Web Services Journal**. *He is a respected industry expert and a consltant with a leading Internet service company.*
SEAN@SYS-CON.COM

PROVIDING GLOBAL BUSINESS INFORMATION VIA WEB SERVICES

ADSTREET

Meeting a crucial need in the information business

The D&B Global Access Toolkit provides an interface to Java or COM developers for accessing the D&B global database. It utilizes XML for data provisioning and is built on the server side as a 100% Java application.

The Toolkit might not exactly resemble most people's impressions of a Web service. While it relies fully on XML, the COM and Java interfaces provide a simple programmatic API that converts the data into XML. Developers can create standalone applications in Java, VB, or C++ that can access the Web services. For example, online markets frequently need to qualify credit worthiness or new market participants. Using the Global Access Toolkit, such a marketplace could qualify a new applicant member company as part of the initial enrollment process. Rather than waiting weeks to be granted trading status, such status could be granted immediately upon receiving a positive rating from D&B. Commercial lending could also work the Toolkit into their applications, possibly enabling mobile professionals to grant small business loans instantly based upon real-time access to the rating information from D&B.

CYNTHIA HAMBURGER, CTO, Dun & Bradstreet

At the time the Toolkit was conceived, many of the emerging standards of Web services, such as SOAP and UDDI, were still in the specification phase. The D&B Global Access Toolkit was designed to be compliant with as many new and emerging standards as possible – it uses the OFX XML standard, for example. D&B is watching the newest standards closely, and the design is modular enough that it would be simple to add SOAP or UDDI to the Toolkit's next release within a matter of weeks.

The D&B Global Access Toolkit provides a typical example of how Web services exist now, and will grow and evolve as the standards progress. It is constructed with the basic building blocks of Web services, but lacks the discovery and introspection capabilities that will be part of all future Web services. D&B has already incorporated XML, Java parsing, EAI, and B2B technologies into the Toolkit. The design was based on the earliest concepts for Web services, namely providing a service with data-neutral technology. Service-based architectures are not a strikingly new concept – many different implementations already exist– but they form the basis of the Web services paradigm when augmented with discovery.

The D&B Global Access Toolkit runs on Windows NT servers and utilizes a SQL Server database to process requests for products. The database acts as a persistent cache for request information and analytics – the actual product data resides in distributed databases across the world. The Toolkit acts as a pipeline to the data residing within these various D&B servers.

The D&B Global Access Toolkit provides six basic services; authentication of the user, authorization for access to specific products, locating the data, building the product, charging the account for the product, and delivering the product. A product may be a wide range of information about companies in practically every country in the world, and in almost every language.

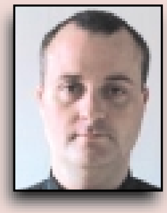
The Toolkit also provides the ability to gather raw data for further analysis. While the scoring and rating of company performance is a typical service, another service provides for the download of data for processing in another system, such as a spreadsheet or financial analysis tool.

The Toolkit's straightforward interface hides the powerful potential of the Toolkit itself. It was designed as a Web-based access and delivery system for D&B data products and services. The initial release provides access to global data products – that is, products that have the broadest customer base. As companies begin to integrate their supply chains via Web services, the need for up-to-date information is crucial in the decision-making process. The D&B Global Access Toolkit provides Web-based access to the D&B global database and is a classic example of how Web services can be deployed on a global scale.

D&B Global Access Toolkit is available for download at http://globalaccess.dnb.com.

Web Services or Peer-to-Peer?

Written by



Oisin Hurley

Oisin Hurley has been involved with distributed objects since 1989. His initial exposure to the field was research into operating system and compiler support for distributed objects. He joined IONA in 1993 to work on the world's first CORBA implementation and since then has worked as a consultant and engineer in the areas of CORBA, J2EE, and Web services. When not developing code, he represents IONA in the OMG, JCP, and W3C standardization activities.

OHURLEY@IONA.COM

Recently I have fielded questions from customers like, "Which is better, peer-to-peer or Web services?" and "Do you think I should move from Web services to a peer-to-peer network?". The answer is another question: "Which approach best suits your solution?". This tends to make the customer think about system requirements a little more, so that once an interaction model is better defined a more informed choice can be made.

This article provides a solution developer with some of the facts about both P2P and Web services so they can make that informed implementation decision.

WEB SERVICES

Most of the current solutions under the title of Web services have a number of elements that are common to their construction. For communication, they use XML to format their messages and deliver them over HTTP. If an interface language is required, another XML syntax called WSDL is used. To deal with the bootstrap issues associated with service and endpoint discovery, clients may access a public UDDI database. This basic framework allows clients to discover and communicate with a service on the public network. Of course, behind the scenes there is an implementation technology that may range from a couple of Java servlets to a full CORBA with services deployment – but the choice is usually constrained by definite factors, such as cost and skill levels.

The characterization of the Web service operation is the classic client/server model. The software fulfilling the role of the server will register with a centralized but replicated datastore. Software that fulfills the role of the client will contact the datastore to discover the server location and can then contact the server. The mechanism to ensure that the client and server can speak intelligibly to each other, or inter-operate, is enforced by well-known standards. This is a straightforward approach to distribution with the advantage that clients are coupled to the servers only by a software contract. Since the contract is fully described, using WSDL for instance, developers can construct clients using the contract information. To set up a stall on the public network, all a developer must do is make a server available at a URL, publish the contract, and advertise.

ENTERPRISE ANALYSIS

Scaling: A large number of clients can swamp a central server, add more servers, implement load-balancing strategies; this may be costly.

Failing: A single point of failure can be catastrophic, add more servers, and implement clustering and standby strategies; this also may be costly.

Security: A single point of data access simplifies access control and authentication.

Integrity: A single logical datastore simplifies enforcing ACID properties.

P2P

Peer-to-peer solutions currently have a different technology usage profile from Web services, but there is no reason why P2P solutions can't use the same XML technologies that Web services use. Centralization of resources is anathema to P2P networks: each participant in the network is equal to every other participant.

The characterization of a P2P network is the peer-to-peer communications model. Each functional unit in the network is *behaviorally similar*. There is no fixed role for a peer as there would be with the client/server distribution paradigm. Peers may do exactly the same thing or there may be transient role assignments. The key element of a P2P system is the peer algorithm, the programmed behavior that makes the system fulfill its intended purpose. The algorithm enforces a logical network topology by defining the concept of "neighboring" peers. The neighbor relationship enables the flow of the essential data in the network, a vital requirement in a system with no central data resource. The fact that the network may be a fluctuating and dynamic one, with peer neighbor relationships breaking and reforming as the load or infrastructure stability changes, is the core attraction of P2P systems.

Unlike Web services, where the client and server roles mean the system deployment requires two separate pieces of software, P2P networks have just one deployable software unit – the peer code. If the peer algorithm can provide for it, P2P networks are easy to extend.

ENTERPRISE ANALYSIS

Scaling: More peers means more work is done cheaper – organizations may deploy peers on many inexpensive machines.

Failing: All peers are behaviorally similar, so there is no functional loss – just make sure that the network algorithm replicates the data.

Security: Difficult unless a peer trust system is established as a simplifying assumption.

Integrity: Can be tricky unless each peer can enforce local ACID properties; lag between nodes may contribute to out-of-date information.

SUMMARY

The Web services versus peer-to-peer comparison is really about two fundamental distributed processing approaches. Each has a network and a way of processing nodes on that network. Indeed, when you look at the details from a node/network abstraction, it's easy to see that a peer-to-peer network that publishes a software contract may actually implement a Web service and similarly a peer-to-peer network may be composed of Web service–style client/server nodes. Choosing an approach is just a matter of matching your requirements with the features each solution provides.

Attunity

www.attunity.com/devx

Reviewed by Joe Mitchko

About the Author:
Joe Mitchko is an independent consultant specializing in Web services and Internet architecture. jmitchko@rcn.com

INTEGRATION SERVER 1.2
by Shinka Technologies

A complete Web services integration platform

No one involved in Web development needs to be reminded of the continued rapid changes in the industry. Even with the current financial downturn, driven by the burst of the Internet bubble, innovation is still happening on many fronts. Nowhere is this truer than in the evolving area of Web services.

Feeling overwhelmed by yet more change, I recently decided to do something I usually never do and subscribed to several Web services–related mailing groups. In this case, some SOAP-related lists would do the trick, especially since SOAP (Simple Object Access Protocol) is one of the key elements of Web service–related architecture. Not long after I signed up, the mail started to trickle in. Actually, it was a flood.

Browsing through several dozen e-mails, I got the sense that implementing the SOAP protocol was not for the faint of heart. Contributors to the list span all levels of expertise from beginners looking for assistance in configuring their SOAP demo environment to open-source code developers making revision announcements. You would also get the occasional frantic message from someone who has had enough but doesn't have a clue how to unsubscribe from the mailing list.

The reason I bring this up is that familiarity with all of the standards that comprise Web service architecture is one thing (SOAP, WSDL, UDDI, etc.); implementing a working Web service is an entirely different subject. In response, a new breed of application servers and IDE-like development tools is hitting the market to address the complexities of e-business integration and drive Web services into the mainstream. One such Web service integration platform is Shinka Integration Server (IS), from Shinka Technologies.

OVERVIEW

The Integration Server suite is available in three editions: Integration Server Java Edition, Integration Server Business Edition, and Integration Server Enterprise Edition. Each edition provides an HTTP-based application server, command center, and designer tool containing one or more adapters. The platform provides you with all you need to add one or more SOAP-based interfaces to your enterprise application. The designer tool helps develop the service interface and automatically generates an XML schema, WSDL service definition, and a test copy of a SOAP request message. The Tomcat-based application server handles secure and non-secure HTTP SOAP requests using dedicated servlets that are specific to your service. The command center runs as a separate HTTP server and provides various administration capabilities including configuration and transaction monitoring.

IS provides you with an XDK library and code generation tools for translating SOAP request (and reply) messages to and from your enterprise application. IS 1.2 not only provides all of the necessary libraries for handling marshalling and dispatching of SOAP requests, it will do a lot of the hard work for you by generating the initial "stubbed out" version of the integration code as well as test SOAP messages.

INSTALLATION

I installed a CD copy of the Business Edition on a Windows-2000 server running on a 1 GHz Athlon machine with 512M of memory without any major difficulties. During installation, you will be asked several questions that are typical of most Java-based product installations, including the JVM directory location, the installation directory, etc. The installation requires JDK 1.3 to be preinstalled, and conveniently provides a copy of the installation on the CD. If you're running an evaluation copy, you must request a temporary license key from Shinka. It arrives via e-mail and requires you to detach an XML formatted file to the configuration directory in the install area.

There were slight glitches during the installation. First, I needed to edit the XDK_ROOT and JAVA_HOME values in the SETENV.BAT file to point to the D: drive and not the C: drive that it appeared to default to. Second, the installation doesn't provide an Oracle JDBC driver, so I needed to place a copy of the CLASSES11.ZIP file in the lib directory and modify the SETCLASSES.BAT file. I had to hunt for a missing PDF file containing the installation instructions (later found on the CD). Then, the user and password I received from Shinka didn't work when I logged into the command center. My hacking skills saved the day as I tried the easy-to-guess admin/admin combination.

You will need to start up two programs through the command menu, the Integration Service and the Command Center. The latter is responsible for serving up the browser-based, administration facility. At this point, we're ready to start designing a Web service.

SHINKA TECHNOLOGIES
1 Market Plaza
Spear Tower, Suite 3600
San Francisco, CA 94105
Tel: 415-293-7817

Web: www.shinkatech.com
E-mail: info@shinkatech.com

AVAILABLE IN THREE EDITIONS
Integration Server Java Edition
Integration Server Business Edition
Integration Server Enterprise Edition

PRICE:
Developer and runtime licenses start at $990 (U.S.). For further information, call Shinka sales.

DEVELOPING A SERVICE

Let's start by developing a simple Web service that will provide online billing and shipment services for company X. Assume the online order system exists behind a corporate firewall. Let's also assume transactions can be made through a series of client calls to an EJB session bean (see Figure 1).

Start up the Shinka Designer tool using the command menu and create a new project.

Next, we need to create an XML schema that will be used by the SOAP interface. The schema will contain a combination of simple and complex XML schema types to define the input and output parameters of SOAP messages. In addition, the XML schema should include data types that define any SOAP exceptions that may be generated by the service. The designer utilizes a UML-based graphical representation to help design your XML schema. You can switch between a graphical representation and an XML-coded (.xsd file) version by switching between the tabs in the designer.

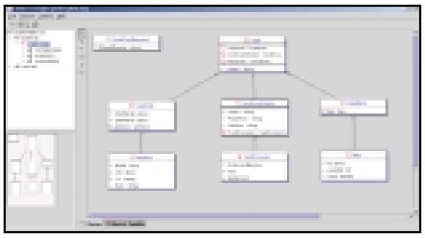


FIGURE 1 XML schema designer

The next step is to create the SOAP service request interface using the designer. You need to provide a unique name for the service along with identifying input, output, and exception parameter names and XML schema types. Again, you can view an XML-formatted view of the interface or the WSDL definition for the service by selecting the tabs on the user interface.

Once you have defined the services for the project, you are ready to generate a series of service stubs. Depending on your platform, the designer can generate Java, C++, or Visual Basic code. (You also may generate enterprise JavaBean components when generating Java code). The stub code that's generated serves as the initial prototype for your implementation, and contains all of the necessary SOAP message conversion bindings to your target language. In fact, the stub is fully functional and can be immediately tested once compiled and configured into the IS server.

DEPLOYING THE SERVICE

The next step is to deploy the service prototype to the Integration Server. This is done by configuring the ISCONFIG.XML file and creating a new service tag pair within the services element of the file. You will also need to make sure that any JAR files created for the service are included in the server's CLASSPATH.

Once you have a service prototype compiled and running on your Integration Server, you can use the designer tool to test your service. The designer will automatically create a test version of a SOAP request message and forward the message to the service. Then you can view the resulting SOAP reply message to assess your service prototype .

Once you are satisfied with the interface, you are ready to migrate the current stub to a fully functioning service. At this point, the designer will be of no further assistance in the continued development of your service (i.e., the tool cannot reverse-engineer existing code). This step requires you to substitute the stubbed-out code for fully functioning client calls to an enterprise application. It's up to the developer to be aware of client requirements and exception handling at this point, and to make sure all exception conditions are handled within the SOAP reply message. In our case, we'd be tying the various SOAP request calls to their corresponding EJB methods using Java. We'd also make sure all exceptions are being properly caught and transposed into SOAP exception element tags.

The beauty of the designer is that it takes care of many SOAP integration issues, including data marshalling (SOAP to native language bindings), service dispatching on the HTTP server, multi-threading, etc.

In addition to generating service-side code, the designer will also create client-side stubs, useful when you need to create SOAP service requests from a remote client.

THE COMMAND CENTER

The command center provides several administration functions that allow you to configure and monitor one or more Integration Servers from your browser (see Figure 2). From the command center, you can monitor the server logs and transactional activity as well as profiling data for each service request.

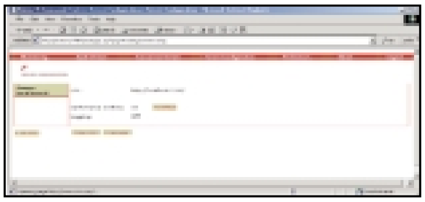


FIGURE 2 Command center

ADDITIONAL FEATURES

- Database logging and audit tracking
- Secure transactions using SSL and HTTPS request headers
- Generates EJB "skeleton" component code based on the Web service description
- Can run standalone (Tomcat server) or on several major application servers, including WebLogic
- Transaction monitoring

FUTURE RELEASES

Some things to look forward to in future releases include SMTP support, Lotus Notes integration, and support for mainframe integration including COBOL code generation and CICS support.

After working with the designer for a while, I noticed improvements that can make it more full-featured. They include adding wizards to help in the service configuration (instead of manually updating an XML file), the ability to edit and compile source code from the designer, and the ability to print from the tool. Since this product and similar ones in the marketplace are still in their infancy, I imagine they may add such features in the future.

DOCUMENTATION

Shinka provides a set of PDF files containing a Getting Started guide and a series of User and Programming guides. Getting Started is especially helpful to those new to Web service development. It provides a good starting point to understand the technology, and for those interested in additional information the guides include several Web site links.

OBTAINING A COPY

You can get a free one-month evaluation copy of Integration Server Version 1.2 by going to the Shinka Web site at www.shinkatech.com/licenses/license.phtml. You must fill out a simple registration form prior to downloading the installation package. You'll also need to sit tight and wait patiently for your license key via e-mail.

CONCLUSION

The Shinka Integration Server and designer tools take care of many initial setup tasks involved in creating a Web service by generating the XML schema, WSDL definitions, and code generation for binding SOAP messages to your platform's target programming language. Command center features, including user management and database logging, are useful when administering the Integration Server in a production environment.

Handling a lot of the tedious work involved in developing and setting up a SOAP-based Web service, Shinka Integration Server version 1.2 leaves you with time to do more important things – like checking your e-mail.

Written by Norbert Mikula

Groundhog Day

XML's binding brackets don't change the rules of business, says WSJ's Industry Editor

So here we are, more than four years since XML (eXtensible Markup Language) first saw the light of the public day. We have come a long way since the early days.

The XML hype started with a small group of experts who argued that SGML wasn't suitable for Web-based publishing and that HTML is pure evil anyway. Today, top-level executives of larger and smaller companies around the world announce that XML will solve all the problems of the world, including, but not limited to, the common cold.

At this point, XML found its place – if perhaps only temporarily – next to all the other initialisms beloved of business people, such as ROI, TCO, and more. "What the heck. It has three letters, like many of the popular acronyms, so it must be good."

I recall in 1994 laughing over a comic that showed a CEO telling his poor IT executive, "I want a Web strategy and I want you to tell me why I want one." Today it appears to me that we are at the same place with XML.

As I followed the exciting evolution of XML, I had the opportunity to become both participant and bystander as history started to unfold. What have I seen? Let me share my two cents worth of wisdom.

'TOT GESAGTE LEBEN LAENGER'

Which loosely translated means, "What's been pronounced dead lives longer." Here are a few examples:

The biggest fear (or wish) since 1996: that **XML will replace HTML**. Sure, why not? It seems to make sense to replace old angle brackets with sexy new angle brackets. Well, certainly it would make sense to replace HTML with a true data-format that can be parsed and processed without having to write megabytes of parsing and heuristic algorithms. However, we are still talking about one being a language to present information, the other being a language to describe it.

XHTML, of course, is a completely different beast. Finally, we have well-formed HTML combined with a powerful and established formatting language. Heaven at last? Let's wait and see. After all, we are in 2001 and HTML is still with us and deployed widely and wildly.

Another big fear (or wish) since 1996: that **XML will replace EDI**. Many would argue that it makes sense to replace archaic codes with sexy angle brackets. First, we don't understand them, and second, they're not particularly sexy, they might add.

The reality looks somewhat different. If I'm a channel master and dictate what formats my suppliers have to use, why bother changing what I am doing? "If it ain't broken don't fix it." Pointy brackets don't change the rules of business in the same way that the Internet does not change easily established business models – as we are ever so painfully reminded when we look at our investment portfolios.

EDI is alive and growing. That said, it's certainly not going away any time soon.

Interesting things happen, however, when we take two disciplines and collectively benefit from the lessons learned, and ebXML is one such example. Here we have people who bring to the table the experience of decades of national and international electronic trade combined with the "new age" Internet and markup technology experts and visionaries.

MONEY, MARKETING, MASS HYSTERIA

XML was grassroots, no doubt about it. There was a time when you could put all of the world's XML experts into a small room and every meeting would start and end with a big group hug (sometimes metaphorically, sometimes literally).

Things change when marketing folks and evangelists realize that there is something that sells a great vision. Things change when something is a great means to secure a new round of financing or potentially lift your stock price (way back when). Things change when the big boys enter the picture and endorse this new thing as The Next Big Thing.

Suddenly everything gets very serious and the stakes rise. Technocrats and visionaries, once enthusiastic "group hug" participants, now engage

Author Bio

Norbert Mikula, who developed the first validating XML parser in Java (NXP) and has been engaged in XML-related efforts since the early days of this standard, is the industry editor of *Web Services Journal*. Norbert currently serves as vice chair of the board of directors of OASIS, the Organization for the Advancement of Structured Information Standards. The chief technology strategist of Washington-based DataChannel, he is the author of numerous white papers and articles and has been a speaker and chair at a variety of national and international conferences and industry events.

NORBERT@DATACHANNEL.COM

Altoweb

www.altoweb.com

in a game of visionary tug-of-war. Not only do companies start competing in winning over the hearts and checkbooks of customers, but they also compete in vision and glorified statements.

A certain amount of mass hysteria is actually very beneficial for advancing standards and technologies. Experts from different disciplines apply their expertise and energy to this new thing and bring forth new and exciting solutions to solve new and exciting problems.

As with so many other grassroots initiatives, once something enters the mainstream and has to live up to the expectations of large scale corporate deployment and support, it inevitably has to lose some of its “innocence.”

THE XML RUBBER MEETS THE ROAD

The XML hype has subsided. Today we are actually seeing deployment of XML outside of textbooks and conference proceedings. We’re also seeing articles that question whether the whole thing was a good idea to begin with. This kind of skepticism is good, as it clears the air and lets us refocus on what’s real.

These negative reflections come, for a good reason, from the people in the trenches, those that have to solve today’s very real business problems. They see most clearly that sometimes a chasm exists between visions and the realities of life.

It’s always entertaining to see somebody dare to attack an established sacred cow and question it. Hordes of faithful followers are immediately at hand to defend the “unfair and unreasonable” attack, using every trick in the rhetoric book.

Sign of the times: Industry celebrities are now talking about “what works and what doesn’t.” Large conference organizers are selecting themes for this year that dispel the XML myth, like: “XML: What Really Works?”

We have gotten past the craze to soberness and from now on we can build solutions and solve real business problems, using XML where it makes sense to do so. Now that the worker bees are taking over, we can expect the evangelists to focus their attention on the new Next Big Thing. It’s time to get down to business; XML has entered production mode.

ENTER WEB SERVICES – THE NEW NEXT BIG THING

So, what happens to XML evangelists? Are we standing in the food lines alongside the dot-com casualties? Not at all. While the XML worker-bees are taking XML to the next level, evangelists have moved on to the next big thing: Web services.

“Web services give XML something to do,” (to abuse a popular tag line of yesteryear).

Web services is a new standards-based approach to an old thing – enabling distributed computing over the Web. Just like XML many years ago, Web services was a grassroots initiative promoted and kept alive by a small group of devoted technologists and visionaries.

Technology vendors today have identified Web services, a.k.a. the “RPC via the Internet” as the missing piece of the puzzle to their solutions portfolio. What we can expect to see in the ensuing months is the same craze around Web services that we saw with XML in its glory days of hype.

I also find some of the statements regarding the new business models amusing. Will we see the emergence of new ways of doing business? Certainly! However, let's not forget that for many years to come there will still be companies that work with a handful of selected customers and that will not need dynamic negotiation of (Web) services via registries. For many years to come, there will still be channel masters that dictate how one has to conduct business and by what means.

There will be one place, however, where I believe Web services may have enormous impact on the business communities and that is the integration of small to medium enterprises into a global e-business community.

SO WHO WILL GET KILLED THIS TIME?

2001 marks Groundhog Day, when, just like XML before it, the Web services initiative will see its technology adoption cycle realized, and whole Web services will see the light of day.

Well, for one it is programming and programmers. Since Web services will bring about higher degrees of reusability we’ll end up knitting together new applications out of existing components. The businessperson will become the developer.

As I pointed out recently on a panel at an XML conference – and as was pointed out to me many panels ago, doesn’t this remind us of the promises made by 4GL sortware packages? Doesn’t it remind us of the promises associated with object-oriented technologies?

WHERE DO WE GO FROM HERE?

I sincerely believe we have an exciting future ahead with Web services. We’ll see a certain degree of hype, followed by a healthy dose of reality and disillusionment. At the moment, the Web services hype has gone far ahead of implementation reality. So we have some catching up to do.

Roll up your sleeves: Web services has entered the marketplace.

Borland

www.borland.com

The Business Tra

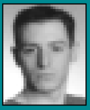
By their very nature, Web services operate in a loosely-coupled, geographically-dispersed environment. From an infrastructure perspective, what does this mean for transaction processing systems? Do the existing approaches to handling transactions through the use of an XA-compliant, two-phase commit transaction manager apply directly to Web services? Typical transaction management infrastructures have complete control over the resources that participate in a transaction: either every resource fully commits or fully rolls back.

In a Web services environment, however, the resources that need to participate in a business transaction need to be aligned, but not dictated to. The business actions of one service need to coordinate and align themselves with the business actions of another service. Even though each node or participant in a Web service can have an independent workflow that executes on its own business rules, the entire business transaction still needs to behave as a single unit. In a loosely-coupled environment, there can't be a central transaction manager dictating all of the rules, commits, and rollbacks. In many circumstances, the business rules of an individual participant may override the intentions of the business transaction. In a business world, transactions need to be coordinated with oversight while still providing voting privileges for the participants. Additionally, just because some participants may not favor the outcome of the business transaction and/or vote against it, the transaction may still continue to commit.

In a business environment, this is analogous to any negotiation. For example, when an author negotiates a new book contract, that author negotiates the terms and arrangements with multiple publishers. Some publishers may offer higher royalties while others may commit senior editors to oversee a project. The author and each of the publishers need to participate in a single business transaction. In this scenario, as long as the author and at least one publisher are still participating, the business transaction can continue. The business transaction could still fully commit even though some of the publishers opt to roll back their individual participation. In an XA-compliant, two-phase commit transaction scenario, the flagging of rollback from any resource eventually causes the entire transaction to roll back.

As another example, a car manufacturer may only want a business transaction to fully commit if they can purchase 100,000 tires for less than $1,000,000 and subcontract the painting and detailing of their cars for less than $500 per car. If one of these conditions can't be met by either trading partner, then the entire transaction needs to roll back. To commit to delivering 100,000 tires, the tire manufacturer needs to lock up the resources necessary to produce the tires. Additionally, for the painting subcontractor to commit to the transaction, they need to ensure availability of supplies. At any given time, any participant can back out of the negotiation and the business transaction. And, based on the demands and rules of engagement, the transaction can continue or be cancelled.

WRITTEN BY
Tyler Jewell



Author Bio:
Tyler Jewell is BEA's Principal Technology Evangelist. Tyler is the coauthor of *Mastering Enterprise JavaBeans* 2.0, coauthor of *Professional Java Server Programming (J2EE 1.3)*, a regular J2EE columnist at http://www.onjava.com>www.onjava.com, and the technoology advisor to theserverside.com where he writes about Web Services.
TYLER@BEA.COM

nsaction Protocol

A critical infrastructure component

Businesses naturally run this way, and a technology infra-structure to support the same type of behavior is needed. A technology infrastructure is required to link business actions into a cohesive unit across departmental or corporate boundaries. This fall the OASIS Business Transactions technical committee (www.oasis-open.org/committees/business-transactions/) will adopt the Business Transaction Protocol (BTP) as an OASIS Committee Specification. The BTP extends the two-phase commit transaction management approach to address the needs of disparate trading partners that use XML to exchange data. This article provides some basic information about the BTP. Architects, developers, and businesses expecting to perform intra-enterprise or trading partner integration will find the BTP a critical component for their future projects.

THE BASICS OF BUSINESS TRANSACTIONS

At the lowest levels of the BTP, there are two types of work orders: atoms and cohesions. An atom is the simplest unit of work and behaves like existing XA-compliant, two-phase commit transactions. An atom of work follows the traditional ACID (Atomic, Consistent, Isolated, Durable) properties and must either fully commit or fully roll back. In a Web Services environment, the operations exposed by a single Web service and/or the internal processes of that Web service would usually make up a single atom. It's possible to have an atom that spans Web services, but there are many limitations to allowing that.

A cohesion is a set of atoms that can be manipulated by a business transaction's initiator. With a cohesion, an initiator is allowed to dictate whether each atom within the cohesion succeeds or fails, even if the atom is capable of succeeding. A cohesion is a transaction that is run by a voting/enrollment process where the initiator of the transaction has the final approval or rejection vote. The initiator can apply business rules to its decision-making process in full light of the recommendations made by all of the atoms in the transaction. Another way to view a cohesion is as a reduction process: when a cohesion begins, many atoms can be enlisted as part of the business transaction, but the cohesion provides a structured approach to reducing the choices available, with the intention of driving a single, successful outcome in the end.

A cohesion can easily be structured to be run by a program. For example, the sample code shown in Listing 1 demonstrates what creating an atom and cohesion may look like. Note that the BTP doesn't define a standard API and that this is purely for demonstration purposes.

THE ROLES IN A BUSINESS TRANSACTION

The BTP specification defines its scope based upon the actors that can be involved with a business transaction. There are a variety of roles defined in the specification:

- ***Initiator:*** This is a software agent that initiates a business transaction. The initiator sends application messages to a Web service in order to invoke operations. The service responds to the initiator in kind. The initiator's organization is considered the Party and the Web service's organization is considered the Counterparty. In a business transaction, there can be a single Party and many Counterparties, each of which contain mult-iple Web services.
- ***Coordinator***: This is a software agent that can decide the outcome of a single atom. The coordinator tracks the set of participants that are enrolled in a single atom and has the same

lifetime as an atom. A coordinator instructs participants to prepare, cancel, and/or confirm. The coordinator makes its decisions based upon input from participants and the initiator.

• ***Participant:*** This is a software agent that is capable of executing prepare, cancel, and/or confirm commands issued by a coordinator. The participant has a designated business transaction protocol communication address where communication with the coordinator occurs. A participant is responsible for executing the termination protocol (i.e., if a confirm command is sent, then the results of all actions of the services contained within the participant must be made permanent; if a cancel command is sent then the actions of the services must be undone).

• ***Service:*** A software agent that handles and responds to application messages. A service participates in a business transaction by way of a participant.

Figure 1 shows the relationships that the different actors can have. Even though there are four different relationships that can occur, the BTP specification only defines the coordination protocol between the Coordinator and the Participant and the operation invocation protocol between the Initiator and the Service. The definition of the demarcation and participant API is left to other bodies such as the Java Community Process. The code snippet provided in this article is a simple example of a demarcation API that may be created for cohesions and atoms.

It's important to understand that the operation invocation protocol isn't a competitor or likeness of SOAP. Rather, it defines extensions to existing XML-based interoperability standards to facilitate invocation operations under the context of an atomic or cohesive business transaction. Transaction identifiers and other context information about entities participating in a business transaction must be relayed on every message passed between two processes. The operation invocation protocol defines the extensions necessary to facilitate this behavior.

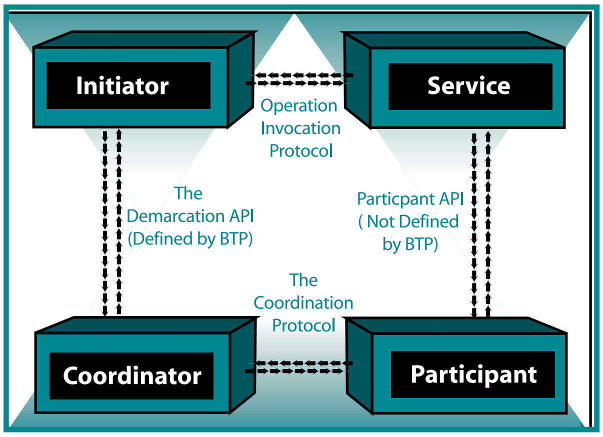

Initiator
Service
Operation Invocation Protocol
The Demarcation API (Defined by BTP)
Particpant API (Not Defined by BTP)
The Coordination Protocol
Coordinator
Participant

FIGURE 1 **Actor relationships**

UNDERSTANDING THE PROTOCOL

The BTP is a protocol, but doesn't necessarily participate in a protocol stack. TCP/IP, RMI, DCOM, and SOAP are all protocols that operate as part of the wire protocol stack. They dictate the semantics of conversation between two nodes that are invoking services. The BTP isn't part of this protocol stack and should be considered an infrastructure support mechanism. (If the wire protocol were drawn as a top-down stack, the BTP would be a dotted line accessory.) You can liken BTP to other infrastructure technologies that are aligned with wire protocols, but aren't part of the stack, such as security, routing, reliability, and attachments.

Additionally, the BTP will occur behind the scenes from any developer. Developers will use a demarcation API, such as something similar to the Java Transaction Architecture (that hasn't yet been built) to control an Initiator of a business transaction, but not to create the BTP protocol messages. The BTP protocol messages will be sent between Initiator/Service and Coordinator/Participant transparently as the state of the system progresses.

Figure 2 shows a big-picture view of the actors and the invocations that they can make upon one another. There are a variety of invocations that can be done as part of the BTP, most of which take place behind the scenes in a typical application. The specification defines its scope based upon the actors that can be involved with a business transaction.

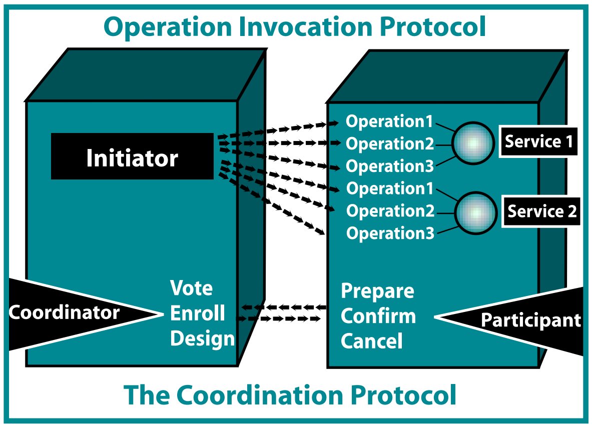


FIGURE 2 **Actors and invocations**

There are a variety of messages that are communicated between actors:

• ***Operations:*** An initiator invokes the operations of a service using a BTP-supported operation invocation protocol. The BTP operation invocation protocol could work over SOAP or the ebXML TRP, for example.

• ***prepare():*** This message is sent from a coordinator to a participant. A participant returns successfully if the set of operations that's participating in the atom is capable of successfully canceling or confirming. It's similar to the first phase of a two-phase commit.

• ***cancel():*** Sent from a coordinator to a participant, this message instructs a participant to process a countereffect for the current effect of a set of procedures. It's essentially the rollback of an atom.

• ***confirm():*** This message is sent from a coordinator to a participant, and instructs a participant to ensure that the effect of a set of procedures is made permanent. It's the commit of an atom.

• ***vote():*** This message is sent from a participant to a coordinator. It can be either unsolicited or in response to a previous prepare() message. A participant can vote to cancel, ready, ready with inability to cancel after timeout, or ready with cancel after timeout. The participant uses these messages to inform the coordinator of the current and upcoming status of the participant.

• ***enroll():*** This message is sent from a participant to a coordinator when a participant has a set of services that want to participate in an atom. When an initiator first makes an invocation on a remote service, the invocation will have the atom identifier associated with it. The service will relay that identifier to a local participant that will contact the coordinator and ask to enroll in the atom.

• ***resign():*** This message is sent from a participant to a coordinator when the operations on the service have had no effect on the atom. It informs the coordinator that the service should no longer participate in the atom.

WHERE TO GET THE PROTOCOL

Currently, there aren't any infrastructures that support BTP, but it should rapidly be adopted in the upcoming year. BEA Systems, HP, Choreology, Interwoven, AppliedTheory, IPNet, Bowstreet, Entrust, Sun Microsystems, and Talking Blocks are all members of the technical committee. Any of these vendors could provide a BTP infrastructure in the very near future. For example, BEA Systems, Inc. is already looking at how BTP will be incorporated into a near-release of the WebLogic Integration platform. Also, upon acceptance of the BTP within OASIS, Microsoft and the Java Community Process will likely follow suit by forming committees to create standard demarcation and participant APIs. Additionally, Choreology currently has a simple trading demo based upon the BTP. It can be downloaded from www.choreology.com/.

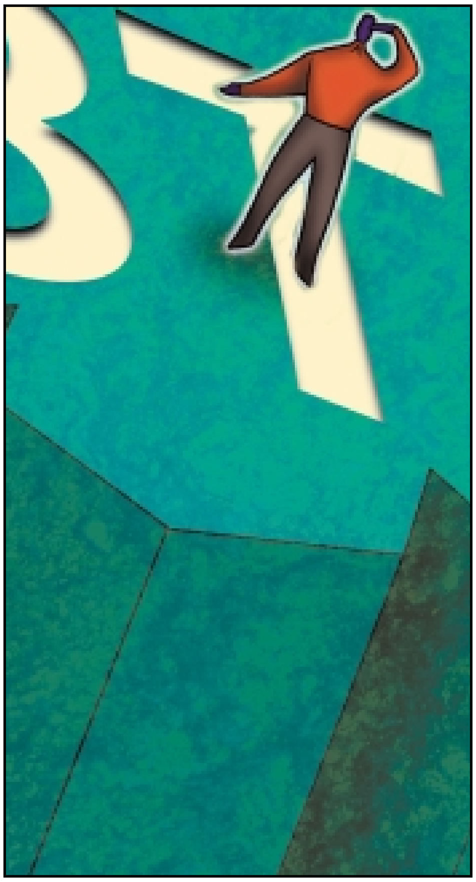
SUMMARY

The BTP specification provides a much-needed addition to the infrastructure community. Its support among transaction managers will provide flexibility not currently available within XA-compliant, two-phase commit transaction engines. The BTP specification provides participant autonomy so that the decisions of one participant mimic the behavior of a corporate negotiation. Corporate negotiation requires compen-sation-based reversals, coord-ination of cohesions, and participant-defined timeouts that can impact an atom, but not necessarily a cohesion. Additionally, the BTP allows for a transaction to operate with discontinuous services that may come and go. In a real organization, quotes and contract negot-iations can last milliseconds or years; the BTP allows for long-running transactions without limiting the scalability of any single participant (a participant is allowed to cancel based upon a timeout dependent on its business requirements). Also, the BTP is designed to be interoperable via XML across any number of communication wire protocols, allowing it to work with an application built on any popular platform.

For more information about the BTP or to see the latest draft, visit the OASIS Web site at www.oasis-open.org.

Listing 1

```
void cohesion()
  {
       // We only need one supplier with the best quote.
Atom supplier1 = new Atom();
Atom supplier2 = new Atom();
Atom supplier3 = new Atom();

// Get quotes from each supplier here (not listed).

// After getting quotes from each supplier, we need to make sure that they
// can commit with the appropriate resources.

supplier1.prepare();  // If no exception, they can fully commit.
supplier2.prepare();  // ditto
supplier3.prepare();  // ditto

// Do business logic here to determine which supplier should be used.
// Let's assume that supplier 2 wins the contract.
//
// The cohesion must be completed by letting each supplier know their
// individual outcome.

supplier1.cancel();
supplier2.confirm();
supplier3.cancel();
    }
```



WSJ INTERVIEW

SilverStream eXtend™: TALKING WITH A WEB SERVICES LEADER

Web Services Journal recently caught up with Steve Benfield, chief technology officer of SilverStream Software, for a quick look at SilverStream's new Web services environment, SilverStream eXtend.

WSJ: Why do you work at SilverStream?

Benfield: I accepted the CTO role at SilverStream because I believe in our vision—simplify and accelerate the development of enterprise-class Java and Web services applications. For 17 years I've helped companies build and implement systems, and I've taught architecture to thousands of developers. I bring a real-world perspective and I think I can help IT organizations build great systems. That's why I work here.

WSJ: You've got a new product, SilverStream eXtend. What is it?

Benfield: SilverStream eXtend is the first integrated services environment that runs on top of J2EE application servers. You use SilverStream eXtend to create services from Java, XML-enable existing legacy systems, assemble services into business processes, and create complex portal, JSP, and client applications that use those services. We tie everything together with a set of intuitive and easy-to-use tools. Finally, everything is deployed to J2EE servers such as BEA WebLogic, IBM WebSphere, or our own SilverStream eXtend Application Server.

WSJ: What do you think sets SilverStream eXtend apart from other products?

Benfield: SilverStream has always been about making complex things easier for developers. Late last year we asked ourselves: What will people really want to do with Web services and how can we make it easy?

Early on we knew we had to go way beyond basic Web services. We view Web services as an enabling technology—just part of what needs to be done. Application developers really won't care about getting into the low-level mechanics of Web services—they just want to build functional applications.

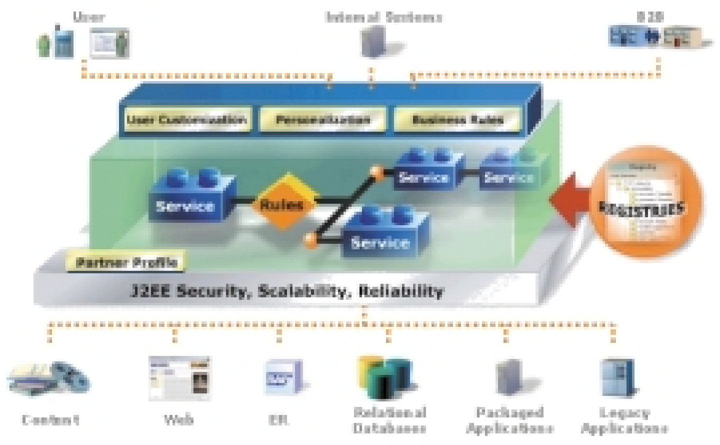
We see two major branches of Web services development: building services and using services. It sounds basic but it led us to package the product in two pieces: eXtend Composer for building services, and eXtend Director for using services.

First I'll describe eXtend Composer: How do you orchestrate services? How do you build and assemble them? Another word is services syndication.

The core functionality needed to build most e-business applications already exists in current production or legacy systems. So besides the easy stuff—creating Web services out of Java code—we also make it easy to visually build Web services out of legacy systems such as EDI, CICS, 3270/5250 streams, MQ Series, SQL, and Telnet sessions. We are the only ones doing that today. We also have a WSFL-compliant business process engine and GUI for assembling services. Assembly becomes really important now because people need to develop things faster and faster and the less hard-coding the better. In addition to assembling services through XML transformations and business rules, we allow Java developers to get at every point of the process. It's the best of both worlds—automation and programmability.

Steve Benfield CTO, SilverStream

SilverStream eXtend Complete Architecture Overview



Once you've built services, you've got to use them. Here's where eXtend Director comes in. Director is two things: an advanced portal and a development framework. eXtend Director includes services such as personalization, workflow, business rules, content management, security, device transcoding, and user profiling. It addresses two audiences: people that want to quickly create portals without a lot of coding and people that want to create applications using JSP, EJBs, servlets, or client code but need personalization or other services. As a developer, you can use Director in three ways: the GUI's we provide to build and manage portals and portal components, APIs and tag libraries for Java development, and Web services for building apps that don't run on the server itself. For example, you could build a Microsoft .NET application that uses our personalization engine. With SilverStream eXtend, integrated apps with Java on the server and Microsoft on the client are a real possibility. The architecture is very elegant.

Finally, SilverStream eXtend Workbench unifies all the GUIs for our product and makes deploying applications very easy. You can use the IDE of your choice with our stuff and deploy everything in one step.

WSJ: Who should look at SilverStream eXtend?

Benfield: Anyone who has to build enterprise class Java or Web services applications should look at our products. BEA or WebSphere architects should check us out because we run really well on those servers. If you want to experiment with Web services, you should download our stuff at http://extend.silverstream.com.

WSJ: When will all of this be available?

Benfield: By the time this is published we'll be in full beta. Release date is late October. eighty percent of SilverStream eXtend involves updating existing products so most of the functionality has already been battle tested.

Internet World Wireless

www.iw.com

Written by Mark Moore

Sending Out-of-Band Messages to SOAP-Based Web Services

Removing some of the programming drudgery

SOAP, the Simple Object Access Protocol, is a lightweight toolkit for building Web services. It is an amalgam of ubiquitous technologies – HTTP and XML. Though the likes of Microsoft, IBM, and the Apache Software Foundation normally have little in common, all support it as a foundation for deploying Web services. One of the great advantages of SOAP's lightweight nature is the simplicity of server-side programming. A SOAP service needs no knowledge of the SOAP environment. In fact, just about any Java class that exposes public methods can be turned into a SOAP service.

Unfortunately, sophistication usually brings complication. While there is little you need to do to write a SOAP-based Web service, it is difficult for a service to know much about the context of a request being made. Should the service evolve over time, you might want the client to provide a service version so a proper response may be sent to a back-revision client. If a service is available to a number of applications, it may be useful to know which is making a request. The obvious approach to solving this problem is to add additional parameters for these things to the service. This quickly becomes annoying for the service programmer, as every method in every service would require these parameters. Things become tedious for the client programmer also, as such information is relatively static and would be duplicated across every call.

A BASIC SOAP WEB SERVICE

Consider Listing 1 (all code listings may be found at www.sys-con.com/webservices/sourcec.cfm). TestService certainly is a trivial service, but no special coding is required to make this a SOAP service. The SOAP RPC router and Java reflection do the work of receiving a request and matching it up with a deployment descriptor and method signature. The code to call it would look like Listing 2. You will need a copy of SOAP from Apache to try the examples. These were built with version 2.0. Use the SOAP administrative interface to deploy the service.

The obvious way to add information like application name and service version to this example would be to add parameters to the call. A better approach would be to develop an out-of-band communication channel with SOAP services. Likewise, the service would need some way of finding these values while servicing a request. As HTTP is one of the component technologies, can the additional information be included in HTTP headers?

HTTP AND SOAP

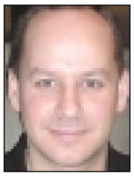
Looking at the conversation between the example client and server above, we see the conversation shown in Listing 3. With the exception that the content of the POST is a SOAP payload, there is nothing unusual about this interaction. So, if we are to use HTTP headers to transmit out-of-band messages, how can we affect the SOAP conversation to include additional headers? It might be tempting to use the SOAPAction value, but its definition is somewhat fuzzy and probably should be avoided. The mechanism we should use is not obvious from the client example above, but there is a way.

EXTENDING THE SOAPHTTPCONNECTION

There is a method on the Call object that enables us to provide an instance of a SOAPTransport. SOAPTransport is responsible for transmitting the SOAP payload to the server and receiving the response. Normally, this would be an instance of SOAPHTTP Connection, though you could use an SMTP-based transport. Looking at the documentation for SOAPTransport, we see that the send() method enables us to contribute headers to the request. There is also a getHeaders() method that enables us to see the headers from the response. Our solution lies in extending SOAPHTTPConnection and providing our own send(), as in Listing 4.

Our send() arranges for the out-of-band messages to be included with any headers SOAP has inserted, then calls super.send(). The messages are stored in a static hash table, so they will be shared across every instance of this SOAPTransport. We have also included some helper methods to add, remove, and get messages from the hash table. To use this new SOAPTransport we modify TestMain as in Listing 5.

The static initialization block will ensure that any messages are set on our SOAPTra-nsport before we have a chance to use it. If TestMain was a servlet, the APP_NAME and APP_VER values might be obtained from the servlet configuration and set during the servlet's init(). Running the modified



Author Bio

Mark Moore is currently an independend consultant. Previously, he was chief architect for KPMG International's Global Knowledge Exchange, deploying knowledge-sharing capabilities to 80,000 KPMG employees in more than 40 countries. He has been designing and developing Web-based knowledge management and collaborative solutions for the last six years.

SINANJU@MEDIAONE.NET

sample, the message sent from the client now looks like Listing 6.

CREATING A SERVICE CONTEXT FOR SOAP SERVICES

The headers are now included in the request being sent to the server, and the server is accepting them, though not acting upon them. On the receiving end, the challenge is the fact that the SOAP service is running in a context-free environment. If we can solve our problem without building a dependency upon SOAP mechanisms into our services, we will be able to use our services in other environments. Listing 7 shows a class that provides the necessary context.

Note the use of an InheritableThreadLocal to store a hash table. Values stored with a ThreadLocal's get() and set() methods are unique to that thread. An Inheritable ThreadLocal ensures that any threads created after a value is set inherit that value. This is important since, server-side, the request is driven by a servlet called RPCRouterServlet. Most servlet containers, such as Tomcat or JRun, will create a pool of threads to service requests. Each time we make a request, a thread is taken from the pool and returned when the request is complete. Our service may be called by any number of applications (and versions of applications) and, using an InheritableThreadLocal, the application name and version may be attached to the thread servicing the request.

Next, we must extend the SOAP RPCRouterServlet as in Listing 8. This extension intercepts the call to doPost() (RPCRouterServlet rejects GET requests), extracts the messages, and adds them to the ServiceContext. Following the call to super.doPost(), it calls ServiceContext.clear() to ensure that the thread does not service another request with incorrect information attached to it. The final step, Listing 9, updates the service to use the ServiceContext. Now, running the TestMain class:

```
> java TestMain foo
```

The returned value was 'The argument you sent was 'foo', your application name is 'TestMain', your application version is '1.0''

CAUTIONS

The idea of using a ThreadLocal or InheritableThreadLocal client-side to transmit user credentials (i.e., value of Http ServletRequest.getRemoteUser()) might be tempting. Doing so asks the client of a service to identify itself honestly. Sending a username and password would be done in the clear. Unless you secure access to your services to trusted clients or have trusted clients digitally sign the message, it probably isn't a good idea.

If you look at SOAPSMTPConnection you'll see that, although it does implement the SOAPTransport interface, it ignores the headers parameter on the send() method. This technique will not work with an unmodified Apache SOAP implementation if you are using SMTP as a transport.

SUMMARY

Using HTTP headers as an out-of-band communication channel is relatively straightforward, and the channel can just as easily be extended to be bi-directional. Use of this technique should take some unnecessary drudgery out of SOAP programming.

The eXtensibility

by Ali Solehdin

Let's focus on the evolution of the Simple Object Access Protocol from its XML roots. In particular, I discuss how XML serves as the impetus for SOAP, and how its extensible nature can be used to effectively serialize data for distributed messaging. Let's begin by creating a simple XML messaging structure, then augment this example incrementally to illustrate how this generic XML messaging model can naturally evolve to conform to the SOAP specification, thereby punctuating the "Simple" in SOAP and "eXtensible" in XML.

BUILDING BLOCKS

To begin the derivation of SOAP from its first principles, let's consider the following scenario. If an online merchant wishes to expose a method to place orders for an item, the retailer could use a PlaceOrder method to accept relevant information from the client. In this instance, information such as the item name, item number, quantity, customer name, credit card type, and credit card number are all relevant to conducting the transaction. Once this information is processed and an order has been placed successfully, the method should return an order confirmation number for tracking purposes.

Such a method can be captured by the following prototype:

```
OrderNumber =
PlaceOrder (ItemName,
ItemNumber, Quantity,
CustomerName,
CreditCardType,
CreditCardNumber)
```

Author Bio

Ali Solehdin is a software engineer in Enterprise Architecture and Advanced Technologies at Infowave Software Inc., a leader in developing wireless applications for the enterprise. He has worked extensively in developing XML-based applications and infrastructures, particularly with SOAP, to develop distributed, firewall-friendly frameworks. Solehdin is a strong proponent of Web services, XML, and their family of related technologies. asolehdin@infowave.com.

If this method resides locally, consuming this method is a straightforward process. However, because this method resides on a remote server, a client can't simply issue a call to PlaceOrder to consume this method. Instead, the client must serialize the method call into a predetermined format, transport that serialized call across the network, and have the remote server deserialize the call, locate the remote object, and execute the method call. Once the call has been executed, the remote server must serialize the return value and send this result back to the client for interpretation. Figure 1 illustrates this process.

In addition to serializing the method call, there must be a means for transporting the call across the network. Although we could perform serialization and transport using a variety of mechanisms (DCOM and CORBA, for example, use their own specific wire and transport protocols), our objective is to use open, platform-neutral standards to realize this goal. So, let's serialize this method call using XML and transport the call across the network over one of the most pervasive transport protocols in use today: HTTP.

By serializing the call using XML and transporting it over HTTP, the request can be sent to a Web server listening for traffic on Port 80. A server-side script could act as a listener and dispatcher to interpret and parse the data stream and invoke the remote method.

One possible approach to serializing method calls using XML is to represent the method name as a parent element, and represent the parameters using child elements. Taking this approach, we can serialize the PlaceOrder method as follows:

```
<PlaceOrder>
    <ItemName>Lightsaver</ItemName>
    <ItemNumber>21382</ItemNumber>
    <Quantity>2</Quantity>
    <CustomerName>Luke
Skywalker</CustomerName>
<CreditCardType>Visa</CreditCardType>
<CreditCardNumber>1234567890</CreditCard
Number>
</PlaceOrder>
```

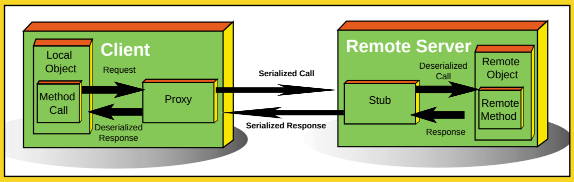


FIGURE 1 **The PlaceOrder process**

This serialization can be performed by a client proxy and sent to a remote endpoint using the HTTP POST method. On the server side, a stub will be required to handle the POST request, deserialize the XML stream, activate the requested object, and execute the method call (Figure 1). Once the requested method has been executed and the return value obtained, the server stub must serialize the response back to the client. One possible representation of the response could take the following form:

```
<PlaceOrderResponse>
    <OrderNumber>10029645</OrderNumber>
fl Return Value
</PlaceOrderResponse>
```

In this case, we have created a parent node by concatenating the method name with "Response" and embedded the return value as a child of this node. While this isn't the only means of serializing the return value, it's a straightforward and effective format.

Although the above approach achieves the goal of serializing a method call in XML for remote execution, it is somewhat limited. For instance, some calls to a remote object may wish to provide additional processing information, such as the name of a second merchant to forward the request to, in the event that the requested item is out of stock. However, the above implementation by itself doesn't provide this capability because it's inappropriate to include this information with the method call. Fortunately, the extensible nature of XML makes it possible for us to augment the existing message structure to include additional details.

If we follow the approach formulated by HTTP requests, the client proxy could include additional information pertaining to routing, security, or debugging into a header, and separate the method call from this header. We can achieve this decoupling by wrapping the headers inside a <Header> element and encapsulating the method name inside another element such as <Body>. Since more than one parent node is now being added to the XML stream, we need an all-encompassing root element to ensure that the XML document is well formed. This root element can also be named arbitrarily, but since it envelops the entire XML structure, it is appropriate to call it the <Envelope> node. Our XML message structure now takes the form shown in Listing 1.

We have now successfully serialized the method call for remote execution by separating the method payload from processing details to provide us with additional flexibility. This flexibility is especially valuable in the Web services arena, whereby vendors and retailers can collaborate with each other to provide value-added services to customers.

AUGMENTING THE MESSAGING MODEL

Since XML is extensible, the preceding serialization can be further modified to support compound data structures. For instance, instead of receiving the parameters ItemName, ItemNumber, Quantity, CustomerName, CreditCardType, and CreditCardNumber as simple data types, the PlaceOrder method could receive compound types, which separate the item and customer details.

For example, details regarding the item to purchase can be encapsulated by the following data structure:

```
Struct ItemInfo
{
    char*   ItemName;
    int     ItemNumber;
    int     Quantity;
}  ItemInfo;
```

Similarly, customer information can be captured in the following CustomerInfo structure:

```
Struct CustomerInfo
{
    char*    CustomerName;
    char*    CreditCardType;
    int      CreditCardNumber;
}  CustomerInfo;
```

With the introduction of the ItemInfo and CustomerInfo structures, we need to update the method interface:

```
OrderNumber = PlaceOrder (ItemInfo,
CustomerInfo)
```

The next step requires us to serialize this augmented method call in XML. Because of the extensible nature of XML, one means of achieving this serialization is to represent the compound type as a parent node and its embedded simple types as child nodes (see Listing 2).

We can now begin to see how the extensibility of XML can be used effectively to help us serialize richer data types.

NAMESPACE ADDITIONS

Another point to consider in this XML serialization is that of namespace collisions. We should distinguish standard keywords we have defined above (Envelope, Header, Body) through a unique namespace qualification. Furthermore, it would be useful to distinguish our PlaceOrder method from other methods with the same name. Since XML supports namespaces, the above representation can be easily augmented to conform to the namespace format:

```
NamespaceIdentifier: ElementName
xmlns:NameSpaceIdentifier=URI
```

Let's start by namespace-qualifying our root element (Envelope) and all nodes directly below it (Body, Header). Since we are at liberty to define our namespace, let's qualify these keywords with the namespace identifier "SOAP-ENV" referenced by the URL http://schemas.xmlsoap.org/soap/envelope. In addition, we can identify our PlaceOrder method with the namespace identifier MyMethod, and the corresponding URL http://myserver.com. A similar format can be applied to the AddRoute header sub-element. Note that we only need to namespace-qualify the PlaceOrder method name and not the parameters, as the parameters are scoped to the method's namespace.

By incorporating the above namespace declarations, we have the XML serialization shown in Listing 3.

Take a close look at the serialized method call in Listing 3 because it represents a valid SOAP message. We have naturally progressed from taking a local method call, extending it to invoke a remote method, using XML to serialize the method call, and augmenting this message model to incorporate additional requirements. In the process, we have leveraged the flexibility of XML and observed how this extensibility can be used to arrive at a messaging model conforming to the SOAP specification.

SERIALIZING ADDITIONAL COMPLEX STRUCTURES

Because of the extensibility of XML, we can further expand upon this messaging model to support additional requirements while still conforming to the SOAP specification.

For example, if a customer wishes to create an electronic wallet of payment methods, we can capture this requirement by extending our XML data structure within the SOAP message. Let's say the customer currently has three credit cards whose balances are all close to their maximum limit, and that he plans to apply for additional credit cards routinely. The interest he pays on these cards varies, so he would like to use his low interest card as much as possible. He'd prefer to avoid the hassle of checking balances on his card prior to each transaction, and would rather defer that decision to the online merchant. Furthermore, he'd like the flexibility of adding additional cards to his wallet dynamically and easily.

While there are many ways to represent this electronic wallet, for illustration purposes let's represent it as a singly linked list. That is, we want to create a linked list of credit card information, which is sent to a processing server, which begins at the head of the list (lowest interest credit card) and traverses the list until a credit card with a sufficient balance is found. Alternatively, the merchant server could split the bill among a number of low interest cards as a value-added service. (This type of service is another representation of Web services in action, with dynamic collaboration between a retail service and a credit card service).

We could represent this linked list as illustrated in Figure 2. Based on this representation, each node of the linked list may be defined as follows:

```
typedef struct PaymentNode
{
   char*   CreditCardType;
   int     CreditCardNumber;
   struct PaymentNode* pNextNode;
};
```

By defining this data structure to represent a list of credit cards, we can update the PlaceOrder method interface and define pHead as a pointer to the head of the list:

```
int PlaceOrder (ItemInfo, CustomerName,
PaymentNode* pHead)
```

The next challenge is to serialize this linked list in XML. Using the extensible nature of XML, one way to achieve this goal is to use the ID attribute to uniquely reference each node of the linked list. Because we also need a means of identifying the head of the linked list, let's define an attribute called root to designate the first node in the list. Similarly, we can define an attribute called null to indicate the end of the list. Using these extensions, we could serialize our linked list in the following fashion:

Notice how the root attribute references the head of the list. This attribute is actually a valid SOAP construct (scoped to the SOAP-ENV namespace), and can be used by an element to broadcast itself as the *root* of a graph. The above serialization achieves our goal of capturing a linked list in XML, and we can incorporate it into our existing SOAP message, which now takes the following form:

Notice how the call in Listing 5 is serialized. A reference to the head of the list is created inside the body of the PlaceOrder method, but the list itself is defined outside the PlaceOrder method.

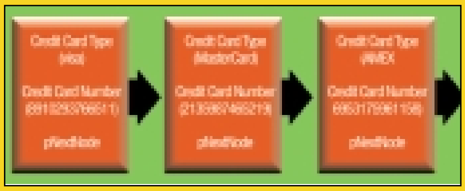


FIGURE 2 The linked list

WHAT ABOUT DATATYPE BINDING?

Note that so far our datatypes have been serialized as invariants, without any type declarations on our part. We can, however, explicitly define our data types in one of two ways. In both cases, we inherit simple types from the "XML Schemas Specification 2: Datatypes."

First, we could create an element schema to define types, in which case we achieve early binding. For example, we could define our ItemInfo and CustomerInfo data structures in an element schema as follows shown in Listing 6.

An alternative approach is to declare types in line with serialization using the xsi:type attribute. In this case, we achieve a late bound model, enabling a polymorphic element (accessor) to access values of several types at runtime. For example, we can take our first iteration of serializing PlaceOrder and bind each parameter inline to a type as seen in Listing 7.

Here, we have declared the basic types set forth by the XML Schemas Specification, and created instances of the type inline during serialization, to bind our elements to simple data types.

NEXT STEPS

By progressing through a natural sequence, we have derived a complete SOAP messaging structure to conduct distributed messaging between a local client and a remote server. We began with a local method call and extended it to invoke a method on a remote server by serializing the call using XML. In the process, we observed how the extensibility of XML allowed us to augment our messaging model incrementally and arrive at the SOAP specification through a natural progression. We also saw how to expand upon this model to accommodate additional requirements and obtain greater flexibility. SOAP provides us with a simple, standardized means of conducting distributed messaging. We now have a better understanding of the basis for SOAP, and how its simplicity is derived from the extensible nature of XML.

Of course, SOAP provides many additional capabilities beyond what was described here, but many of those ideas can be derived from the first principles using the same exercises we have performed in this article.

I encourage you to formulate some of these ideas for yourself, as it will help you develop and extend your own SOAP applications. Keep in mind that the extensibility of XML must be handled with care; the extensible nature of XML can be a danger in and of itself, opening the door for a myriad of messaging structures to emerge. This is why standardization efforts such as SOAP are so important, to lay the groundwork for a well-understood and common messaging format. ⓔ

Listing 1

```
<Envelope> fl Root element for well-formed XML
<Header>
   <AddRoute>http://mysecondfavoriteretailer.com</AddRoute>
</Header>
<Body>
   <PlaceOrder>
      <ItemName>Lightsaver</ItemName>
      <ItemNumber>21382</ItemNumber>
      <Quantity>2</Quantity>
      <CustomerName>Luke Skywalker</CustomerName>
      <CreditCardType>Visa</CreditCardType>
      <CreditCardNumber>1234567890</CreditCardNumber>
   </PlaceOrder>
</Body>
</Envelope>
```

Listing 2

```
<Envelope>
<Header>
   <AddRoute>http://mysecondfavoriteretailer.com</AddRoute>
</Header>
<Body>
   <PlaceOrder>
      <ItemInfo>            fl Compound Type
         <ItemName>Lightsaver</ItemName>
         <ItemNumber>21382</ItemNumber>
         <Quantity>2</Quantity>
      </ItemInfo>
      <CustomerInfo>         fl Compound Type
         <CustomerName>Luke Skywalker</CustomerName>
         <CreditCardType>Visa</CreditCardType>
         <CreditCardNumber>1234567890</CreditCardNumber>
      </CustomerInfo>
   </PlaceOrder>
</Body>
</Envelope>
```

Listing 3

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope">
<SOAP-ENV:Header>
   <myRetailers:AddRetailer xmlns:myRetailers="http://
myserver.com/retailers">
      http://www.mysecondfavoriteretailer.com
   </myRetailers:AddRetailer>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <MyMethod:PlaceOrderxmlns:MyMethod="http://
myserver.com">
      <ItemInfo>
         <ItemName>Lightsaver</ItemName>
         <ItemNumber>21382</ItemNumber>
         <Quantity>2</Quantity>
      </ItemInfo>
      <CustomerInfo>
         <CustomerName>Luke Skywalker</CustomerName>
         <CreditCardType>Visa</CreditCardType>
         <CreditCardNumber>1234567890</CreditCardNumber>
      </CustomerInfo>
   </MyMethod:PlaceOrder>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4

```
<pHead href="#Card1" />    ⬅ Reference to head of list

<PaymentNode id="Card1" root="1">    ⬅ Define head of list
   <CreditCardType>Visa</CreditCardType>
   <CreditCardNumber>1234567890</CreditCardNumber>
   <pNextNode href="#Card2" />
</PaymentNode>
<PaymentNode id="Card2">
   <CreditCardType>MasterCard</CreditCardType>
<CreditCardNumber>0987654321</CreditCardNumber>
   <pNextNode href="#Card3" />
</PaymentNode>
<PaymentNode id="Card3" />
  <CreditCardType>AMEX</CreditCardType>
   <CreditCardNumber>1122334455
</CreditCardNumber>
   <pNextNode null="1" />    ⬅ Designate last node
</PaymentNode>
```

Listing 5

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope">
<SOAP-ENV:Header>
   <myRetailers:AddRetailerxmlns:myRetailers="http://
myserver.com/retailers">
      http://www.mysecondfavoriteretailer.com
   </myRetailers:AddRetailer>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <MyMethod:PlaceOrderxmlns:MyMethod="http://
myserver.com">
      <ItemInfo>
         <ItemName>Lightsaver</ItemName>
         <ItemNumber>21382</ItemNumber>
         <Quantity>2</Quantity>
      </ItemInfo>
      <CustomerName>Luke Skywalker</CustomerName>
      <pHead href="#Card1" />
   </MyMethod:PlaceOrder>
   <MyMethod:PaymentNode id="Card1" SOAP-ENV:root="1">
      <CreditCardType>Visa</CreditCardType>
      <CreditCardNumber>1234567890</CreditCardNumber>
      <pNextNode href="#Card2" />
   </MyMethod:PaymentNode>
   <MyMethod:PaymentNode id="Card2">
      <CreditCardType>MasterCard</CreditCardType>
      <CreditCardNumber>0987654321</CreditCardNumber>
      <pNextNode href="#Card3" />
   </MyMethod:PaymentNode>
   <MyMethod:PaymentNode id="Card3">
      <CreditCardType>AMEX</CreditCardType>
   <CreditCardNumber>1122334455</CreditCardNumber>
      <pNextNode null="1"/>
   </MyMethod:PaymentNode>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
   <xsd:element name="ItemName" type="xsd:string"/>
   <xsd:element name="ItemNumber" type="xsd:int"/>
   <xsd:element name="Quantity" type="xsd:int"/>
   <xsd:element name="CustomerName" type="xsd:string"/>
   <xsd:element name="CreditCardType" type="xsd:string"/>
   <xsd:element name="CreditCardNumber" type="xsd:int"/>

   <xsd:complexType name="ItemInfo">
      <xsd:sequence>
         <xsd:element ref="ItemName"/>
         <xsd:element ref="ItemNumber"/>
         <xsd:element ref="Quantity"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="CustomerInfo">
      <xsd:sequence>
         <xsd:element ref="CustomerName"/>
         <xsd:element ref="CreditCardType"/>
         <xsd:element ref="CreditCardNumber"/>
      </xsd:sequence>
   </xsd:complexType>

</xsd:schema>
```

Listing 7

```
<SOAP-ENV:Envelope xmlns:SOAPENV="http://
schemas.xmlsoap.org/soap/envelope"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-Instance">
<SOAP-ENV:Header>
   <myRetailers:AddRetailerxmlns:myRetailers="http://
myserver.com/retailers">
      http://www.mysecondfavoriteretailer.com
   </myRetailers:AddRetailer>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <MyMethod:PlaceOrderxmlns:MyMethod="http://
myserver.com">
      <ItemName xsi:type="xsd:string">Lightsaver</ItemName>
      <ItemNumber xsi:type="xsd:int">21382</ItemNumber>
      <Quantity xsi:type="xsd:int">2</Quantity>
      <CustomerName xsi:type="xsd:string">Luke
Skywalker</CustomerName>
  <CreditCardTypexsi:type="xsd:string">Visa</CreditCardType>
      <CreditCardNumberxsi:type="xsd:int">1234567890
</CreditCardNumber>
   </MyMethod:PlaceOrder>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Special Introductory Offer SAVE $13.89*

WebServices JOURNAL

The world's leading independent Web Services technology resource

Get Up to Speed with the Fourth Wave in Software Development

- Real-World Web Services: Is It Really XML's Killer App?
- Demystifying ebXML: A Plain-English Introduction
- Authentication, Authorization and Auditing: Securing Web Services
- Wireless: Enable Your WAP Projects for Web Services
- The Web Services Marketplace: An Overview of Tools, Engines and Servers
- Building Wireless Applications with Web Services
- How to Develop and Market Your Web Services
- Integrating XML in a Web Services Environment
- Real-World UDDI
- WSDL: Definitions and Network Endpoints
- Implementing SOAP-Compliant Apps
- Deploying EJBs in a Web Services Environment
- Swing-Compliant Web Services
- and much, much more!

Only $69.99 for 1 year (12 issues)
Newsstand price $83.88 for 1 year

**Offer subject to change without notice*

PREMIERE ISSUE

WebServices JOURNAL

www.wsj2.com

SYS-CON Events presents

From the Editor by Jack Martin pg. 5

Guest Editorial by Ajit Sagar pg. 7

Case Study by Robert Diamond pg. 70

Success Stories by Brady Flowers pg. 82

Tips & Techniques by Sean Rhody pg. 94

WSJ News pg. 98

Interview by Robert Diamond pg. 104

IMHO by Jason Westra pg. 110

Feature: Is It Really XML's Killer App? Real-world Web Services are here to stay — Brady Flowers 22

Security: Authentication, Authorization and Auditing Support for any kind of e-business — Jack Martin 30

Wireless: Enable Your Web Services with WAP Finding common ground — Sean Rhody 38

Tutorial: How to Develop and Market Your Web Services A means for generating more traffic — Ajit Sagar 46

WSDL: Definitions and Network Endpoints Open standards-based technology makes the difference — Israel Hilerio 54

Feature: Deploying EJB in a Web Services Environment Run on any platform — Robert Diamond 62

Web Services and Swing: An Introduction High-performance translations — Jason Westra

Feature: Implementing SOAP-Compliant Applications Exploring technology

SYS-CON MEDIA

www.wsj2.com

SYS-CON MEDIA

SYS-CON Media, the world's leading publisher of i-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of Web Services.

Written by Mani Chandy, Ph.D.

Bridging the Web Services ---- Divide -----

Bringing perception and implementation together

Web services offer a widely-accepted standardized framework for objects to discover and interact with each other using ubiquitous technologies: XML over HTTP. This article is about bridging two "divides" in discussions of Web services in the press and newsgroups. The first divide is one of perception: ubiquitous services offer radically new ways of thinking about distributed systems, whereas much of the discussion today is about limited incremental steps from today's Web-page technology. The second divide is economic and social: some companies have the resources and the desire to adopt Web services rapidly while others do not. This article explores solutions to cross both divides.

WEB SERVICES: RADICAL TECHNOLOGIES OR INCREMENTAL STEPS?

The initial adoption of a new technology is often incremental. The first cars looked like horse-drawn buggies. The massive impact of a technology is not felt until later.

Web services are radical disruptive technologies. But, just as with the automobile, the initial discourse about Web services is about incremental steps.

First, let me explain why the collection of Web services is a radical technology. Then, I'll discuss ways to harness the technology.

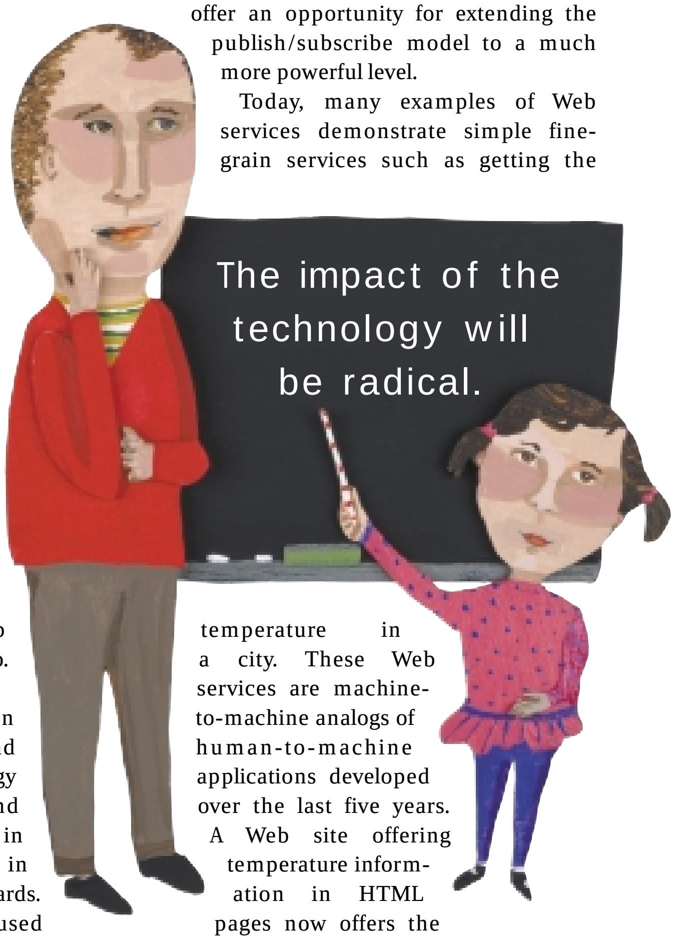
Distributed systems based on message passing or remote procedure calls have been discussed for decades, as have directories that allow objects to find other objects. The theoretical foundation for Web services was established years ago. So, why is the technology radical?

First, Web services are based on ubiquitous technologies: XML and HTTP. Second, major technology companies such as IBM and Microsoft have taken the lead in defining the standards and in developing tools for these standards. The combination of widely-used technologies and intensive support from leading technology companies will result in the rapid spread of Web services standards.

PASSIVE WEB SERVICES VS ACTIVE WEB SERVICES

The most popular examples of Web services showcase passive services. A passive service responds to a Web service request. For example, a currency trading service responds with the dollar-to-yen conversion ratio to a request that specifies dollars and yens. Likewise, a weather service responds with the temperature in Barcelona when a request is made that specifies Barcelona.

Active services offer additional power. They are proactive, whereas passive services are reactive. Listeners register with an active service and the active service invokes a listener with information for which the listener subscribes. Message-queuing systems, such as IBM's MQ Series and Tibco's TIB, demonstrate the value of the publish/subscribe model. On a different scale, the JavaBeans model also showcases the advantages of listeners subscribing for events. The service concept in general and Web services in particular offer an opportunity for extending the publish/subscribe model to a much more powerful level.



Today, many examples of Web services demonstrate simple fine-grain services such as getting the temperature in a city. These Web services are machine-to-machine analogs of human-to-machine applications developed over the last five years. A Web site offering temperature information in HTML pages now offers the same information as a Web service.

Web services can offer more value by providing coarse-grain complex services that are composed of many fine-grain components. Certainly, returning the list of movies playing in a town is a useful service, but an even more valuable service is returning the number of integrated circuit chips of a specified type that can be delivered to a specified factory from suppliers around the world. The latter service is a composition of smaller services that determine how many chips each supplier can deliver and by what date, and then integrate the

Author Bio

Mani Chandy, Ph.D., is chief scientist and cofounder of iSpheres Corporation (www.ispheres.com), a pioneer of meta application frameworks which enable companies to manage business in real-time. A renowned authority on distributed computing, Dr. Chandy is Simon Ramo professor of computer science at the California Institute of Technology, and a member of the National Academy of Engineering.

MCHANDY@ISPHERES.COM

WebServices JOURNAL

home
advertise
subscribe
contact

SYS-CON MEDIA

Dot.com
- buyer's guide
- xml forums
- mailing list
- xml jobs
- xml store

Magazine
- advertise
- authors
- customer service
- editorial board
- subscribe

Content
- archives
- digital edition
- editorial
- features
- interviews
- product reviews
- source code

Conferences
- web services edge
santa clara, ca
oct 22–25

e search

Search

e specials

Collect All Three!

JDJStore.com
$175.99

JDJ, XML, CFDJ - ALL THREE!!! - The Complete Works

e best sellers

1. RoboHELP Office 2000 from eHelp
$899.99

2. MultiActive: ecBuilder Pro v6 for Win32
$204.99

3. Archos Portable 6GB MP3 Player / USB Hard Drive
$349.99

4. Garmin: StreetPilot III GPS
$569.99

5. V-Communications: Partition Commander V6.0
$39.99

What's Online

www.sys-con.com/webservices

Join the fourth wave of technology and become part of the newest paradigm in software development!

Welcome to WSJList!

Welcome to WSJList!

Web Services Journal proudly announces WSJList, the NEW Web Services mailing list community at the center of discussions, technical questions, and more...

Welcome to WSJList!

Web Services Journal proudly announces WSJList, the ***NEW*** Web Services mailing list community for discussions, technical questions, and more... Join the WSJList now to monitor the pulse of the Web Services industry!

Web Access:

Subscribe to the WSJList:
Enter your e-mail address below and click subscribe to join the list.

Subscribe

Each week we will be joined in our forum by an industry leader who will take part in discussions on all areas of Web Services. Ask questions of these experts and other developers on the cutting-edge of technology. Get solutions for your development issues and share your knowledge with others.

Join the WSJList now to monitor the pulse of the Web Services industry!

WSJ2.com

Can't get to the newsstand in time? Let www.wsj2.com be your source for industry events and happenings. Check in daily for up-to-the-minute news, events, and developments, and be the first to know what's going on in the latest paradigm of technology.

Digital Edition

Don't have your print edition on hand? can't wait for the next issue to arrive in the mail? Our digital edition is just what you need. As long as you have your computer with you, you can read ***Web Services Journal*** anytime, anywhere.

Web Services Edge 2001 International Conference & Expo

Make plans now to be at the Santa Clara Convention Center in Santa Clara, California, October 22-25, 2001, for the largest Web Services conference and expo of the year.

The conference tracks will offer invaluable information on real-world standards, Web Services and Java, EAI with Web Services, and Web Services and XML. Go to www.sys-con.com for updates, news, and registration information.

QUICK POLL

Will Web Services be "The Next Big Thing?"
◯ A- Yes
◯ B- No

Vote

Results

CAPE CLEAR

Click for a FREE 30-day trial
AltoWeb

SilverStream eBusiness Platform
click here
Evaluate the Application Server Now! »»

Shinka Integration Server
shinka technologies

information into a value-added service.

COUPLING ACTIVE WEB SERVICES

Most business scenarios require multiple events that occur in parallel before a business response is necessitated. A supply chain Web service may reveal the following: One application reveals that product demand is skyrocketing. A typical response would be to issue additional production requests to meet the increased demand. However, an inventory management application may indicate an overabundance of product in the channel and increased production would be unnecessary. Either event detected in isolation may not trigger a response, but if aggregated will warrant action – slow down production or face dumping inventory at the end of the quarter.

Developing an active Web service to monitor the extended enterprise for composite events, aggregate and analyze these events, then actuate applications in response, raises the value of Web services as well as the competitiveness of the enterprise.

COMMAND AND CONTROL APPLICATIONS USING WEB SERVICES

Web services offer an opportunity to think about distributed systems, e-business, and collaboration in a different way: command and control mechanisms for the enterprise executive.

The military has used command and control systems for decades. The outcome of wars depends, in part, on the efficacy of C3I (command, control, communications, and intelligence) systems. An Air Force general coined the term "The Warfighter's Infosphere" to mean the command and control system that gives the fighter the information and tools needed, instant by instant. A fighter plane's cockpit is an example of a pilot's infosphere. With today's accelerating pace of business, executives need custom-built command and control systems to monitor and respond to opportunities and threats.

A traditional command and control system consists of three parts: sensors, information fusion and decision support, and actuators. Sensor networks acquire information. Information streams from many sensors are aggregated and decision-support tools are run using the aggregated information. The decisions are fed to actuators that respond to the situation. Radar and sonar are examples of sensors. Computers, databases, and tools for optimizing deployment of military forces are examples of information fusion and decision support. Missiles fired in response to decisions are examples of actuators.

Sensing, fusing, and actuating response to information are equally meaningful to business. Sensors acquire information from Web services, Web-based applications that offer HTML pages, FTP sites, enterprise applications such as supply chain, customer resource management applications, and e-mail. Information fusion and decision-support tools determine whether critical events have occurred and what is the best response to these events. Actuators are interactions with people and applications. For example, the sensor network may show that a supplier will delay shipping a part to a factory. The information fusion and decision-support tools determine whether this delay is a costly disruption of the supply chain and how to respond to the disruption in an optimal fashion. The actuation may be a bid or purchase of an alternative part in an online auction.

Sensing and actuation functions are greatly simplified by widespread adoption of Web services. We can build command and control systems for businesses more easily now because we can build toolkits for developing sensor networks and e-business actuators. A systematic study of command and control systems, coupled with a comprehensive framework and tool set for building them, will produce valuable returns for e-business.

COMMAND AND CONTROL VS TRADITIONAL DISTRIBUTED COMPUTING

The lowest layers of the distributed computing technology stack deal with sending bits on a wire, higher layers deal with sending messages, even higher layers deal with sequences of back-and-forth messages between two parties negotiating a transaction, and command and control applications lie on top of the stack.

A command and control view of an enterprise and a military operation are similar. Just as the commander-in-chief, and commanders of naval fleets, aircraft carriers, and individual aircraft have their own command and control systems, so too do the chief executive officer, general managers of divisions, and sales offices have their own systems. The challenge is to orchestrate multiple units in different theaters so as to achieve the overall goal. This challenge is different from the ones discussed in the context of Web services such as how to make services that quote currency conversion ratios or services that accept bids at auctions. A command and control system is, however, based on sensors and actuators, and obtaining a currency conversion ratio is an example of a sensing, and making a bid at auctions is an example of actuating. We can now build frameworks for command and control systems that run enterprises because Web services take care of standardization of protocols lower in the stack.

JUMP-STARTING THE WEB SERVICES COMMUNITY

When the telephone was invented, it was ushered in with huge fanfare by the technological leaders of the era. At the time, the rest of the world was using telegraph services to communicate. The result was a slower adoption rate of the telephone. The masses needed to be convinced that it was a viable solution worth their attention and investment. A technology gap resulted: on one side resided millions of telegraph users and on the other resided the first telephone users. If the services offered by the telegraph could be extended to comply with the technology requirements of the telephone, the community of telephone users would have grown drastically.

A parallel can be drawn with Web services. Both the leading technology minds and the media are praising its arrival. Yet, until Web services are accepted by the rest of the business world, its adoption rate will be less than what is needed to grow the Web services community. However, if the tens of millions of Web applications and HTML documents can be exposed as Web services, the rapid adoption of Web services by the masses, and the subsequent population of a Web services community, will be ensured.

Leading-edge companies will introduce meaningful Web services applications in the next few years. Major technology companies, including IBM and Microsoft, offer Web services toolkits today. However, many companies will wait until they see a compelling business need to develop new Web services or retrofit their existing Web-based applications as Web services. This is how the Web services divide will take shape unless steps are taken today to bridge the gap. Treating the human-to-machine interaction process as a machine-to-machine Web service is one way to populate the Web services community.

A retail exchange has thousands of suppliers participating from around the globe. A garment manufacturer in Thailand has a Web site featuring the quantity, availability, and price of his wares. Yet he may be unwilling to invest additional resources to convert his Web site to a Web service. The problem appears minor: the Web site was built for human-to-machine interaction, and Web services require that the site be extended for machine-to-machine interactions using Web services standards. But the problem is magnified when multiplied by the tens of millions of Web sites in existence.

One solution is to extract information non-invasively from Web sites and other document repositories by automating the customary steps needed to interact with these sites. This is not standard screen scraping, rather, it is an approach to locate granular pieces of information from within unstructured pages and payloads of documents.

Most businesses have some domain of discourse. For example, denim suppliers talk about color and weave. Different denim suppliers use different terminology, but they all deal with the same domain. Semantic-based systems can use knowledge of particular domains to understand the information within documents that deal with that domain, and extract information accordingly. Semantic-based systems are less brittle than standard screen-scraping, syntax-based systems.

The key business advantage of these systems is that they obtain information non-invasively. The retail exchange in our example extracts information from suppliers and customer Web sites. Suppliers and customers don't need to do anything other than make passwords available to the retail exchange.

Applying this concept to our example, the retail exchange can offer value-added Web services to its members that non-invasively aggregate information from all suppliers and customers. The retail exchange has "Web service–enabled" all its suppliers and customers, without requiring them to do anything.

SUMMARY

IT teams should begin thinking of developing Web services as part of a larger command and control system that resides on the desktops of its executives: an active Web service that senses actionable information, fuses and analyzes its impact, and actuates an orchestration of applications in response.

Bridging the Web services divide is a priority for IT departments. One solution that will help jump-start the adoption of Web services is to non-invasively "Web service–enable" existing applications, without requiring the application providers to do anything.

Web Services & Distributed Component Platforms

PART 1

How They Stack Up

by Boris Lublinsky & Michael Farrell

With the widespread use of component technology, it has become increasingly important to employ components in distributed computing environments. Currently, a handful of distributed component platforms exists, including the Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), and Remote Method Invocation (RMI). A new addition to this list is Web services, a framework that has recently received considerable attention. "It's safe to say that every one of the software platform vendors out there will have to either support, or provide tools for, Web services at some point," said Simon Yates, an analyst with the industry research firm Forrester Research. Although the Web services market is still in its infancy, analysts say it will have a huge bearing on the way future applications are developed and deployed. This article consists of four major parts, the first two of which are presented below; the next two will be covered in the next issue.

The first part examines Web services details, including what Web services are, the Web services Definition Language, and the Universal Description, Discovery, and Integration service. The second part defines fundamental characteristics of distributed architectures, including their basic features, support provided, employment of the distributed service, and inherent issues in the distributed architecture. The third part uses those characteristics to compare Web services with other distributed environments. Finally, the major applications for Web services will be discussed along with Web services' readiness assessment for those tasks.

DESCRIPTION OF WEB SERVICES

WHAT ARE WEB SERVICES?

Web services were introduced as a result of a combined effort by Microsoft, IBM, and Ariba. Web services are components accessible from standard Internet protocols. They combine the best aspects of component-based development and the Web. Like components, Web services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web services are not accessed via object model–specific protocols, such as DCOM, RMI, or IIOP. Instead, Web services are accessed via ubiquitous Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML). Furthermore, a Web Service interface is defined strictly in terms of the messages the Web service accepts and generates. Consumers of Web services can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web services interface.

Several essential activities must occur in any environment that supports Web services:

- A Web service needs to be created, and its interfaces as well as invocation methods must be defined.
- A Web service needs to be published to one or more intranet or Internet repositories for potential users to locate.
- A Web service needs to be located in order to be invoked by potential users.
- A Web service needs to be invoked to provide any benefit.
- A Web service may need to be unpublished when it is no longer available or needed.

Web services architecture requires three fundamental operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find required services using a service broker and bind to them.

To perform these three operations and bind in an interoperable manner, there must be a Web services stack that embraces standards at each level. Figure 1 shows a conceptual Web services stack. The upper layers build upon the capabilities provided by the lower layers.

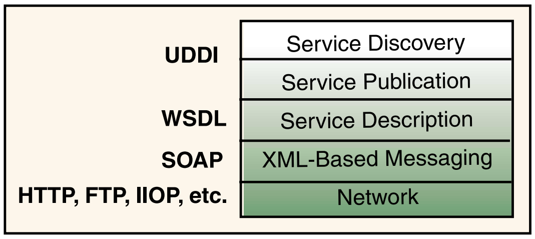
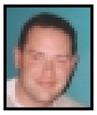


FIGURE 1 Conceptual Web servicestack

Author Bios

Boris Lublinsky is a Regional Director of Technology for the Central Region at Inventa, where he oversees engagements in EAI, B2B integration, and component-based development of large-scale Web applications. Previously, he was a Technical Architect at Platinum Technology and SSA, where he developed execution platforms for component-based systems. Boris has over 20 years of experience in software engineering and technical architecture.

BLUBLINSKY@HOTMAIL.COM



Michael Farrell Jr. is a Senior Engineer at Inventa Technologies in Chicago, where he serves as technical leader on B2Bi, EAI, and component-based application development projects. Michael is also a certified trainer and delivers courses on a number of systems integration and enterprise applications. Prior to Inventa, he was a Systems Engineer at Intel Corporation, where he implemented distributed applications to support the semiconductor fabrication process. Michael has two degrees from the College of Engineering at The University of Iowa.

MFARRELL@IOWA.DHS.ORG

Two foundation technologies of Web services are:
- Web Service Description Language
- Universal Description, Discovery, and Integration

WEB SERVICE DESCRIPTION LANGUAGE

Web Service Description Language (WSDL) is an XML-based Interface Definition Language (IDL) for network-based services, operating on messages containing either document-oriented or procedure-oriented information. It is used to specify operations, messages, implementation details, access protocol, and contact endpoints. The operations and messages are described abstractly and then bound to a concrete network protocol and message format to define an endpoint. WSDL is extensible to allow descriptions of endpoints and their messages regardless of what message formats or network protocols are used to communicate; however, the only bindings that exist today describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of data, being exchanged, and port types, which are abstract collections of operations (see Figure 2). The three most important sections in the service definition are:
- Message and Port Type: *What* operations the service provides.
- Binding: *How* the operations are invoked.
- Service: *Where* the service is located.

In addition, any complex data type that the service uses must be defined in a type section immediately before the message section. Let's take a look at each section in more detail.

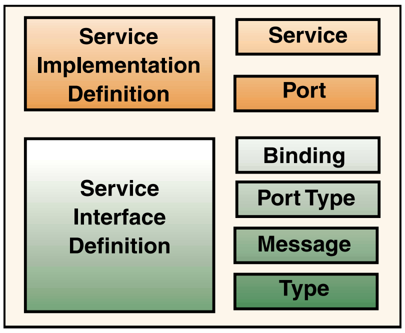

Service Implementation Definition
Service
Port
Service Interface Definition
Binding
Port Type
Message
Type

FIGURE 2 **WSDL definition**

Although the Web services market is still in its infancy, analysts say it will have a huge bearing on the way future applications are developed and deployed.

- ***Types:*** A container for data type definitions using some type system.
- ***Message:*** Corresponds to a single piece of information moving between the invoker and the service. A regular round trip remote method call is modeled as two messages, one for the request and one for the response. Each message can have zero or more parts, and each part can have a name and an optional type. When WSDL describes an object, each part maps to an argument of a method call. If a method returns void, the response is an empty message.
- ***Operation:*** An abstract description of an action supported by the service defining a specific input/output message sequence. The message attribute of each input/output must correspond to the name of a message that was defined earlier. If an operation specifies just input, it is a one-way operation. An output followed by input is a solicit/response operation, and a single input is a notification.
- ***Port Type:*** Corresponds to a set of one or more operations. When WSDL describes an object, each operation maps to a method and each Port Type maps to a Java interface or class.
- ***Binding:*** Corresponds to a Port Type and is implemented using a particular protocol such as SOAP or CORBA. The type attribute of the binding must correspond to the name of a Port Type that was defined earlier. Because WSDL is protocol-neutral, you can specify bindings for SOAP, CORBA, DCOM, and other standard protocols. If a service supports more than one protocol, the WSDL should include a binding for each protocol that it supports.
- ***Port:*** Represents the availability of a particular binding at a specific end-point. The binding attribute of a port must correspond to the name of a binding that was defined earlier. This allows a single component to support multiple communication protocols and effectively be an inter protocol bridge.
- ***Service:*** Modeled as a collection of ports.

Although, as we mentioned above, WSDL is protocol-independent and can support any type of binding, SOAP 1.1 binding is becoming the most popular one today.

ADVERTISEMENT AND DISCOVERY OF WEB SERVICES

As the number of Web services grows, locating a specific service may become a tricky and complex issue. How can a particular service on a particular system be advertised? How can information presented on a particular site be announced to search engines?

The Universal Description, Discovery, and Integration (UDDI) specification defines a way to publish and discover information about Web services. The UDDI specification describes a conceptual cloud of Web services and a programmatic interface that defines a simple framework for describing any kind of Web services. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services. Using the UDDI discovery services, businesses individually register information about the Web services that they expose for use by other businesses.

This information can be added to the UDDI business registry either via a Web site or by using tools that make use of the programmatic service interfaces described in the UDDI Programmer's API Specification. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other on a regular basis. Once a business registers with a single instance of the business registry service, the data is automatically shared with other UDDI root nodes and becomes freely available to anyone who needs to discover what Web services are exposed by a given company.

The core component of the UDDI specification is the UDDI Business Registration, an XML file used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI Business Registration consists of three components: "white pages" including address, contact, and known identifiers; "yellow pages" including industrial categorizations based on standard taxonomies; and "green pages," the technical information about services that are exposed by the business. Green pages include references to specifications for Web services as well as support for pointers to various file- and URL-based discovery mechanisms if required.

The UDDI specification consists of an XML schema for SOAP messages and a description of the UDDI API specification. Together, these form a base information model and interaction framework that provides the ability to publish information about a broad array of Web services. The UDDI XML schema defines four core types of information, which provide the sort of information that a technical person would need to know in order to use a business partner's Web services. These are:

• ***Business information:*** Many partners will need to be able to locate information about your services and will have as starting information a small set of facts about your business, either your business name or perhaps your business name and some key identifiers, as well as optional categorization and contact information. The core XML elements for supporting, publishing, and discovering information about a business – the UDDI Business Registration – are contained in a structure named "businessEntity." This structure serves as the top-level information manager for all information about a particular set of data related to a business unit. The overall businessEntity information includes support for "yellow pages" taxonomies so that searches can be performed to locate businesses that service a particular industry or product category, or who are located within a specific geographic region.

• ***Service information:*** Technical and business descriptions of Web services – the "green pages" data live within substructures of the businessEntity information. Two structures are defined: businessService and bindingTemplate. The businessService structure is a descriptive container that is used to group a series of Web services related to either a business process or a category of services. Examples of business processes that would include related Web services information include purchasing services, shipping services, and other high-level business processes. These businessService information sets can each be further categorized, allowing Web services descriptions to be segmented along combinations of industry, product and service, or geographic category boundaries.

• ***Binding information:*** Within each businessService lives one or more technical Web services descriptions (binding Templates). These contain the relevant data needed by application programs to connect to and then communicate with a remote Web Service. This information includes the address to make contact with a Web service, as well as support for optional information which can be used to describe both hosted services and services that require additional values to be discovered prior to invocation. Additional features are defined that allow for complex routing options such as load balancing.

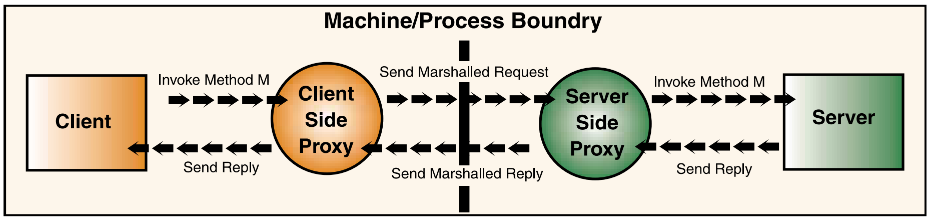


FIGURE 3 Usage of proxies in distributed communications

• ***Service specification:*** Often it is not enough to simply know where to contact a particular Web service. For instance, if it is known that a business partner has a Web service that allows me to send them a purchase order, knowing the URL for that service is not very useful unless a great deal is known about the format the purchase order should be sent in, what protocols are appropriate, what security is required, and what sort of response will result after sending the purchase order. Integrating all parts of two systems that interact via Web services can become quite complex. As an application program or programmer interested in specific Web services, information about compatibility with a given specification is required to make sure that the right Web services are invoked for a particular need. For this reason, each bindingTemplate element contains a special element that is a list of references to information about specifications. Used as an opaque set of identifiers, these references form a technical fingerprint that can be used to recognize a Web service that implements a particular behavior or programming interface.

The UDDI API is divided into two logical parts, the Inquiry API and Publishers' API. The Inquiry API is further divisible into two parts – one part used for constructing programs that let you search and browse information found in a UDDI registry, and another part that is useful in the event that Web service invocations experience failures. Programmers can use the Publishers API to create rich interfaces for tools that interact directly with a UDDI registry, letting a technical person manage the information published inside UDDI registry.

BASIC FEATURES OF DISTRIBUTED SYSTEMS

In order to compare Web services to other distributed systems implementations, we must first define the major characteristics of distributed systems in general.

The basic features of distributed systems are:

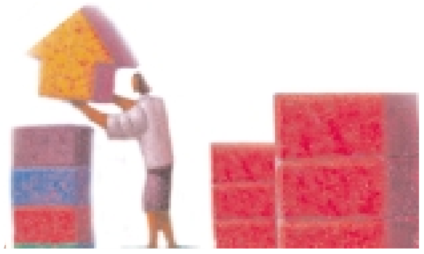
• ***Request and response.*** Because a remote method call is inherently delivered over a network communication infrastructure, it is typically divided into a *request* (asking the service) and a *response* (returning results to the client). In principle, from the client's view, the request and response corresponding to a remote method call can be done as one atomic action (*synchronous call*), or they can be separated, where the client issues the request and, as a future action, issues a wait for the response (*deferred-synchronous call*). Sometimes the response part may be empty (no out parameters and no functional value). In this case, the corresponding method is usually termed a *one-way method*. A one-way method can be called asynchronously, where the client does not have to wait until the call is finished. In a distributed environment the "exactly-once" semantics of remote calls are practically impossible to achieve. Distributed platforms in practice today ensure the "at-most-once" semantics of a synchronous and deferred-synchronous call (exactly-once semantics in case of a successful call, at-most-once semantics otherwise). Best-effort semantics are ensured for a one-way method.

• ***Remote Reference***. One of the key issues of remote method calls lies in referencing remote objects. Classically, in a "local" case, a method call is made as a method invocation on the reference to the target object. However, in a distributed environment we face the issue that an object reference should identify a remote object over the network. It is obvious that classical addresses will not do as the references since they do not contain information about a server in which the target object resides. By convention, a reference that contains all information necessary to access a remote object is termed a *remote reference*. In addition, representation of a remote reference must span the differences in hardware architectures of the nodes where the objects involved in a particular remote method call reside.

• ***IDL interface***. In principle, a client's code and the server object that is subject to a remote call from the client can be implemented in different languages and run on heterogeneous architectures. To accommodate these differences, the interfaces of a server object are specified in an architecture-neutral Interface Definition Language (IDL). Typically, IDL provides constructs for specification of types, interfaces, modules, and (in some cases) object state. However, there is no means for specifyingimplementation. Usually a mapping from IDL to standard programming languages, such as C++, Java, COBOL, etc. is a part of an IDL definition.

• ***Object proxy***. To bridge the conceptual gap between the remote and local style of references, both in the client and server code, the actual manipulation of remote references is typically encapsulated in wrapper-like objects known as client-side and server-side proxies (Figure 3).

The *client-side proxy* and the corresponding *server-side proxy* communicate with each other to transmit requests and responses. Basically, the client-side proxy supports the same interface as the remote object. The key idea behind proxies is that the client calls a method **m** of the client-side proxy to achieve the effect of calling **m** of the remote object.



Thus, the client-side proxy can be considered a local representative of the corresponding remote object. Similarly, the key task of a server-side proxy is to delegate and transform an incoming request into a local call form and to transform the result of the call to a form suitable for transmitting to the client-proxy. Thus, a server-side proxy can be considered as the representative of all potential clients of the remote object.

• ***Marshalling***. Both the request and response of a call are to be converted into a form suitable for transmitting over the network communication infrastructure. Typically, serialization into a byte stream is the technical base of such conversion. By convention, this conversion is referred to as marsh-alling/unmarshalling.

PROVIDING AND EMPLOYING DISTRIBUTED SERVICE

Providing and employing distributed service entails the following:

• ***Registering services with a broker***. In order to be remotely accessible, any service provided by a server needs to be registered with a broker. As a result of the registration operation, the broker creates a remote reference to the service. The remote reference is returned to the server and can be registered with a naming and/or trading service (see below).

• ***Naming***. Because brokers supply remote references in an internal broker format, a distributed object platform typically provides a naming utility to enable the use of ordinary names. A *naming* defines a name space and tools for associating a name with a remote reference. Typical operations supported by a naming service are resolving a name into a remote reference and associating a name with a remote reference.

• ***Trading***. If a client does not know a name of the component it can ask a trading utility (analogous to yellow pages) to provide a list of references to remote services which possess the properties indicated by the client as the search key.

• ***Binding***. As mentioned above, the client can receive a remote reference via naming or trading, or as a result of another remote method call. It is a general rule of almost all distributed object platforms that when a client receives a remote reference to a service, a proxy is created (if it does not already exist) in the client's address space, and the client is provided with reference to the proxy.

• ***Static invocation***. When a client is compiled with the knowledge of the requested service interface (e.g., in the form of an IDL specification), calls made by remote methods can be encoded statically within the client code as calls of the proxy's methods. Static invocation requires recompilation of both client and server in the case of IDL changes.

• ***Dynamic invocation***. In principle, the client and the server can be compiled separately; thus, they may not always be current with respect to the static knowledge of available interfaces. To overcome this obstacle most of the implementations provide broker-hosted metadata about interface and mechanisms for building requests dynamically.

INHERENT ISSUES IN THE DISTRIBUTED ARCHITECTURES

The following are inherent issues in the distributed architectures:

• ***Garbage collection of server objects.*** Server objects not targeted by any remote reference can be disposed and should be handled by a garbage collector. Garbage collection in a distributed environment is a more complex issue, comparing to the single address space.

• ***Transactions***. Transactions are an important tool in making distributed object applications robust. The following consequences of working with distributed objects with respect to transactions should be emphasized. Employing multiple databases is inherent to a distributed environment. This implies that a two-phase commit must be applied. As objects themselves possess state, they can also be considered as resources taking part in transactions.

• ***Concurrency in server objects***. Many server implementations employ multithreading; thus, a server object may be subject to invocation of several of its methods simultaneously. Naturally, synchronization tools have to be applied in the code of the server's objects in order to avoid race conditions, etc. In addition, several threads running on the server may call the broker at the same time, perhaps to register a newly created object.

• ***Reactive programming***. The advantage of easily passing remote references as parameters makes it relatively simple to employ event-based (reactive) programming style without introducing specific callback constructs. Most distributed object platforms define abstractions to support this style of programming.

In this first part of a two-part article, we have defined what Web services are and described their two foundational technologies – WSDL and UDDI. We then defined the fundamental characteristics of distributed architectures in general. Based on these characteristics, in part two of this article we will offer an in-depth comparison of Web services with existing distributed architectures. We will also discuss potential applications for Web services and assess their readiness for real applications.

Written by Jonathan Cortez

Building Web Services Using Microsoft .NET

Harnessing the connectivity of the Internet

Microsoft .NET is the key enabling technology for Microsoft's vision of software as a service. The .NET Framework is the overall infrastructure that provides developers with a platform to create programs that transcend device boundaries and fully harness the connectivity of the Internet.

In this article, I'll illustrate how easy it is to build and use Web services using the .NET Framework SDK. I used the Beta 2 of the .NET Framework SDK and wrote all code samples using C#, Microsoft's new programming language for .NET. This article assumes you have some familiarity with Web services concepts and component-based programming. While I've tried my best to ensure that the technical content of this article is up-to-date, some features and operations in the .NET Framework might change when a newer version of the SDK ships.

To try out the source code in this article, you need to download and install the Microsoft .NET Framework SDK Beta 2 from www.microsoft.com/net/downloads.asp. The SDK includes everything developers need to write, build, test, and deploy .NET Framework applications – documentation, tools, compilers, and samples. You don't need to use Visual Studio .NET to run the code samples in this article. I installed the SDK on my Windows 2000 server machine and used Notepad as my source code editor.

.NET FRAMEWORK OVERVIEW

The .NET Framework is a platform for building, deploying, and running Web services and applications. It provides a highly productive, standards-based, multi-language development environment and runtime infrastructure that simplifies application development in the Windows platform.

The .NET Framework was built from the ground up to meet the needs of Web services developers and consumers. It has pervasive support for XML and SOAP. The general design goals of the NET Framework are to:

- Simplify application development
- Unify programming models across all languages and types of applications
- Utilize Web standards for better interoperability with other platforms
- Make it easier to deploy, version, and maintain applications

As you can see in Figure 1, the .NET Framework is essentially a system application running on top of the operating system, which can be different flavors of Windows and even other operating systems.

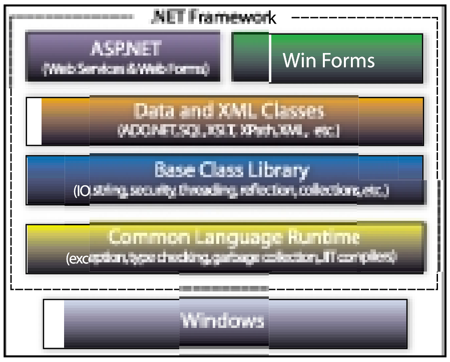

.NET Framework
ASP.NET
(Web Services & Web Forms)
Win Forms
Data and XML Classes
(ADO.NET, SQL, XSLT, XPath, XML, etc.)
Base Class Library
(IO, string, security, threading, reflection, collections, etc.)
Common Language Runtime
(exception, type checking, garbage collection, JIT compiler)
Windows

FIGURE 1 Microsoft .NET Framework architecture

The most important component of the .NET Framework is the Common Language Runtime (CLR). The CLR is a rich runtime environment that provides important system services to .NET objects written in any language that can be represented in the Common Intermediate Language (CIL). The CLR activates objects, performs code access security, lays out objects in memory, verifies type safety, performs garbage collection, and facilitates error handling. Java developers may think of the CLR as the .NET equivalent of the Java Virtual Machine (VM).

Sitting on top of the CLR is the Base Class Library, consisting of classes that support IO functions, string manipulation, security management, threading, reflection functionality, collections, and other functions.

On top of the base class library is a set of classes that provide data and XML manipulation. ADO.NET and SQL classes allow you to access and manipulate persistent data stored on backend databases. The framework also provides a number of classes to manipulate XML, perform XML searches, and perform XML transformations.

Win Forms (or Windows Forms) provides classes for development of native Windows GUI applications. If you've developed MFC applications in the past, you'll fall in love with Win Forms since it supports

Author Bio
Jonathan Cortez is a software architect and technical lead with Cap Gemini Ernst & Young. He develops world-class e-commerce and knowledge management solutions at the company's Advanced Development Center, based in Bellevue, WA. He specializes in .NET and COM+ development using C#, C++/ATL, VB, SQL Server, XML, and other Web technologies. He just completed a project where he led a team in the development of Web services using SOAP for a large online stock brokerage.

JON_CORTEZ@HOTMAIL.COM

easier GUI development and provides a common and consistent interface across all languages.

ASP.NET is the Web application platform that enables developers to easily develop both powerful Web services and use Web Forms to build rich browser-based applications.

SYSTEM.WEB. SERVICES NAMESPACE

The System. Web.Services namespace in the |.NET Framework SDK consists of classes for developing Web services:

- ***WebService :*** Optional base class for classes implementing the Web service. This provides direct access to common ASP.NET objects such as Application and Session.
- ***WebServiceAttribute:*** Attribute used to provide additional information to a Web service such as a description string and a custom XML namespace.
- ***WebMethodAttribute:*** Attribute used to expose a public method of the implementing class to remote Web clients.
- ***WebServiceBindingAttribute:*** Attribute used to declare the binding of one or more Web service methods.

WEB SERVICES IN ASP.NET

ASP.NET provides a development platform that makes it easy for developers to build, publish, and consume Web services. It provides the infrastructure for the inner workings of Web services so developers can focus on implementing the business functionality instead of worrying about the low-level infrastructure details. Within ASP.NET, there are two important pieces of Web services technology: ASP.NET Web services and ASP.NET Web Service Clients.

ASP.NETWeb Services is a technology that allows you to build and expose Web services. Since ASP.NET Web Services are built on top of ASP.NET, you can take advantage of the features of ASP.NET such as caching, state management, and authentication, which are all built into the framework. Also, all ASP.NET codes are compiled rather than interpreted, which offers significant performance improve-ments over the current Active Server Pages (ASP) programming model.

ASP.NET Web Service Clients pertains to any components or applications that reference and make use of a Web service. This reference is made possible through a proxy class for the Web service.

In the following sections, I'll show you how to build and consume a Web service using ASP.NET.

BUILDING A .NET WEB SERVICE

Creating a Web service in ASP.NET is similar to creating any component that exposes programmatic access to its application logic. You typically create a Web service in two steps: Declare the Web service and define the Web service methods.

DECLARE THE WEB SERVICE

In ASP.NET, you declare a Web service by placing the required <% @WebService ... %> directive at the top of a text file with a .asmx filename extension. In this directive, you must specify values for both the Language and Class attributes. The Language attribute tells ASP.NET which compiler to use for the Web service. You can set this attribute to C#, VB, and JS, which refer to C#, Visual Basic.NET, and JScript.NET, respectively. The Class attribute tells ASP.NET which class is implementing the Web service. The implementing class can reside either on the same file or in an assembly. The code example in Listing 1 shows how to set the Class attribute to a class residing in the same file.

The class implementing the Web service doesn't need to reside on the same .asmx file. You can achieve a higher degree of code reusability by placing the implementing class in an assembly. Your .asmx file will contain only the WebService directive, and the assembly needs to reside under the \bin directory of the Web application hosting the Web service. This is illustrated in Listing 2. MyAssembly.cs is compiled with the C# compiler (csc.exe) to MyAssembly.dll using the following command:

```
csc /t:library MyAssembly.cs
```

In the WebService directive in Listing 2, the implementing class name and the assembly name are both specified. If the assembly name is not specified, ASP.NET searches through the list of assemblies in the \bin directory of the Web application the first time the Web service is accessed. Therefore, specifying the assembly name will yield better performance on the first access of the Web service.

Classes implementing Web services can optionally derive from the WebService class of the System.Web.Services namespace. This will allow your implementing class to have access to all the common ASP.NET objects such as Application and Session objects. These common ASP.NET objects can be accessed via the Context property of the WebService class.

Your Web service, by default, will use the XML namespace http://tempuri.org/. I would highly recommend that you specify your own namespace for your Web service before it is made available publicly. To do this, apply the optional WebService attribute to the class implementing your Web service and specify your own namespace in the Namespace attribute. This will distinguish your Web service from other Web services that might be using the default namespace. The code example in Listing 3 illustrates overriding the default XML namespace by specifying a custom XML namespace.

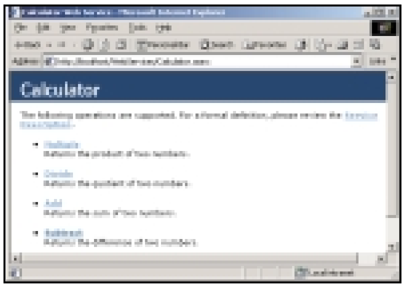
DEFINE THE WEB SERVICE METHODS

By default, public methods of a class implementing your Web service cannot be invoked over the Web. For every method you want to expose to the Web, you need to apply a WebMethod attribute. These methods are called Web Service Methods. You don't need to tag every method in your class as WebMethod unless you want that method to be accessible via the Web. Listing 4 shows a Web Service Method (Hello) that's accessible to Web service clients and a regular method (DoSomeWork) that won't be accessible from the Web.

I created a sample Web service that provides basic mathematical functions such as Addition, Subtraction, Multi-plication, and Division (see Listing 5). The Web service resides in a file called Calculator.asmx. To set up the Web service on my local box, I used IIS 5.0 Internet Services Manager to create a virtual directory called WebServices. I placed the Calculator.asmx file on that directory. The complete URL to this Calculator Web service is http://localhost/ WebServices/ Calculator.asmx. If you deploy this sample Web service on a remote server, you'll need to substitute the name of the server and the virtual directory appropriately.

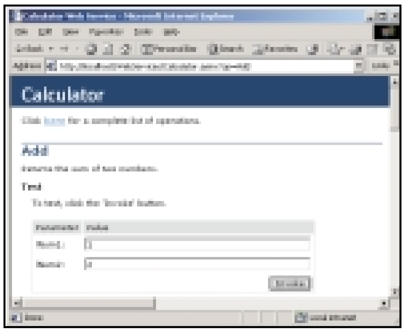
CONSUMING A .NET WEB SERVICE

Clients communicate with Web services via standard Web protocols. Microsoft .NET Web services currently support three protocols: HTTP GET, HTTP POST, and SOAP over HTTP. This means that any application running on any platform should be able to access .NET Web services as long as they are using these standard Web protocols and can understand XML-encoded messages.



Calculator

FIGURE 2 **Calculator Web service page**



Calculator

Add

Test

FIGURE 3 **Default Calculator Web service consumer**

DEFAULT WEB SERVICE CONSUMER

If you point your Web browser to the .asmx file of your Web service, ASP.NET will give you a list of supported methods (see Figure 2). Clicking on one of these methods will bring up a form representing a default Web service consumer (see Figure 3). This default consumer uses the HTTP GET protocol to talk with the Web service and is autogenerated on the fly via .NET reflection. This is a great way to immediately test your Web service methods without writing any client code.

If we try invoking the Add method with 1 and 2 as input parameters, a new browser window is displayed containing the XML-encoded result shown in Figure 4. In this example, the complete URL used to invoke the Add Web method, along with parameters, is the following: http://localhost/WebServices/Calculator.asmx/Add?lNum1=1&lNum2=2

The default consumer also provides the description of how to access the method via SOAP, HTTP GET and HTTP POST. These descriptions help developers understand the encoding of the request and response messages, which is often critical when interoperating with a non-.NET Web service consumer.

WEB SERVICES DESCRIPTION LANGUAGE TOOL

The Microsoft .NET Framework SDK

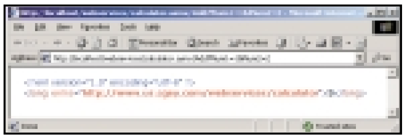


FIGURE 4 **Results page of Web service default consumer**

provides a utility, Web Services Description Language Tool (Wsdl.exe), to generate source code proxies to the actual Web services. This utility uses the WSDL document of the Web service to generate these proxies. The .NET Framework uses reflection to dynamically generate the WSDL document for the Calculator Web service via the URL http://localhost/WebServices/Calculator.asmx?WSDL. The Wsdl.exe tool can also take a WSDL file as its input, instead of a URL pointing to where the WSDL can be obtained. This will apply to scenarios where you'll be generating client proxies against non-.NET Web services.

When you create a proxy class using Wsdl.exe, the default protocol is SOAP. SOAP provides the richest extensibility mechanism of the three protocols supported by ASP.NET in the current release by allowing scenarios where classes and structs can be passed as arguments to the methods of the Web service. If you want to generate client proxy code for the Calculator Web service in the C# language and use the SOAP protocol, you'll need to execute the following in the command line:

```
wsdl /l:CS /o:CalculatorProxy.cs
/protocol:SOAP
http://localhost/WebServices/Calculator.
asmx?WSDL
```

This command will generate a C# source file called CalculatorProxy.cs containing a proxy class, Calculator, that derives from the SoapHttpClientProtocol class.

The /protocol: parameter specifies which protocol you want to use. Since SOAP is the default protocol used by Wsdl.exe, you can omit the /protocol: parameter from the above call to Wsdl.exe. If you want to use the HTTP GET protocol, specify HttpGet. The proxy class generated will derive from the HttpGetClient Protocol class. If you want to use the HTTP POST protocol, you need to specify HttpPost and the proxy class generated will derive from the HttpPostClientProtocol class.

Inspecting the generated proxy class, it includes methods that forward calls to the original Web service methods. You have two choices for how this proxy can be used in your client application. The first option is to include this source file in the client application project. However, the client application project needs to be a C# project. You also need to add into your project any references that the proxy depends on.

A better option for using the proxy class is to compile the C# proxy source file into a dynamic link library (DLL) and then add a reference to this DLL to any client project you want to create. This way, your proxy DLL is self-contained and can be used by different projects implemented in different .NET languages. Below is the command line for compiling the C# proxy source into a DLL (Calculator Proxy.dll):

```
csc /t:library /r:system.web.services.dll
/r:system.xml.dll CalculatorProxy.cs
```

Now let's look at the client application code that will use this proxy DLL (see Listing 6). You compile the client code to a console program (CalculatorClient.exe) using the following command line:

```
csc CalculatorClient.cs
/r:CalculatorProxy.dll
```

No matter how you choose to use the proxy class, the client code will still look the same. First, an instance of the proxy class is created. Then, the appropriate methods of the proxy class are called that will in turn invoke the Web service methods. In this case, the Multiply method was invoked, passing in the value of 4 for both parameters. The resulting value was displayed on the console screen. You should see the following if you run CalculatorClient .exe from the command line:

```
4 X 4 = 16
```

That's all there is to it. Both the proxy class and the .NET Framework do all the dirty work of creating and parsing SOAP messages between the Web service and the client application.

SUMMARY

This article took you on a brief tour of using the Microsoft .NET Framework for building and consuming Web services. As you can see, ASP.NET makes it very easy to write and consume Web services without having to dive into the details of standard specifications such as HTTP, SOAP and WSDL. Web service is a strategic technology in Microsoft's overall vision. This is very evident in the framework's extensive support for Web services and Microsoft's recent announcement of Project Hailstorm – a set of user-centric Web services that includes identification and authentication, e-mail, instant messaging, automated alerts, calendar, address book, and online file storage. I strongly encourage everyone developing in the Microsoft platform to start playing with Microsoft .NET Framework SDK and Visual Studio .NET. Use the information in this article to start writing your own Web services and create programs that consume existing Web services.

For more information on .NET Web services, go to http://msdn.microsoft.com/webservices/. For more general information on Microsoft .NET, visit www.microsoft.com/net/.

Listing 1 **Class contained in the same .asmx file**

```
<% @WebService Language="C#" Class="FooBar" %>
using System.Web.Services;
public class FooBar
{
  [ WebMethod ]
  public string GetMessage()
  {
    return "This class is contained in the same .asmx
file.";
  }
}
```

Listing 2 **Class contained in a separate assembly**

```
In FooBar.asmx:

<% @WebService Language="C#" Class="FooBar,MyAssembly" %>



In MyAssembly.cs:

using System.Web.Services;
public class FooBar
{
  [ WebMethod ]
  public string GetMessage()
  {
    return "This class is contained in a separate assembly.";
  }
}
```

Listing 3 **Overriding the default XML namespace**

```
<% @WebService Language="C#" Class="FooBar" %>
using System.Web.Services;
[ WebService(Namespace="http://www.mycompanyname.com/") ]
public class FooBar
{
  [ WebMethod ]
  public string Hello()
  {
    return "Hello Web Services.";
  }
}
```

Listing 4 **Defining Web service methods**

```
<% @WebService Language="C#" Class="FooBar" %>
using System.Web.Services;
public class FooBar
{
  [ WebMethod(Description="Returns a greeting to the
caller.") ]
  public string Hello()
  {
    return "Hello Web Services.";
  }

  public string DoSomeWork()
  {
    return "I did some work."
  }
}
```

Listing 5 **Calculator Web service (Calculator.asmx)**

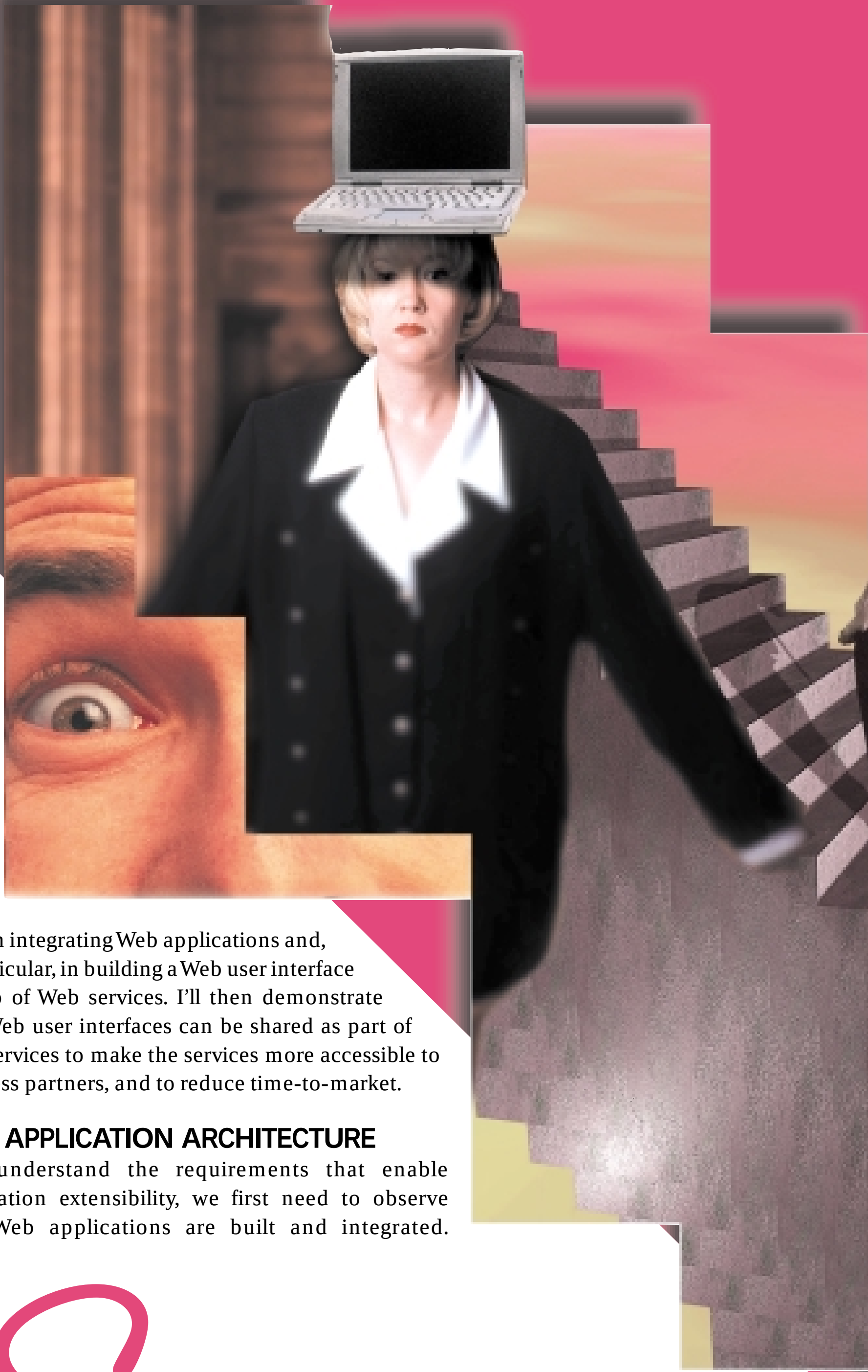
```
<%@ WebService Language="C#" Class="Calculator" %>
using System.Web.Services;
[ WebService(Namespace="http://www.us.cgey.com/webser-
vices/calculator") ]
public class Calculator
{
  [ WebMethod(Description="Returns the sum of two num-
bers.") ]
  public long Add(long lNum1, long lNum2)
  {
    return lNum1 + lNum2;
  }

  [ WebMethod(Description="Returns the difference of two
numbers.") ]
  public long Subtract(long lNum1, long lNum2)
  {
    return lNum1 - lNum2;
  }

  [ WebMethod(Description="Returns the product of two num-
bers.") ]
  public long Multiply(long lNum1, long lNum2)
  {
    return lNum1 * lNum2;
  }

  [ WebMethod(Description="Returns the quotient of two
numbers.") ]
  public long Divide(long lNum1, long lNum2)
  {
    return lNum1 / lNum2;
  }
}
```

Listing 6 **Calculator Web service client (CalculatorClient.cs)**

```
using System;
public class CalculatorClient
{
  public static void Main()
  {
    // Create the proxy
    Calculator oCalcProxy = new Calculator();

    // Invoke the Multiply method and display results on
the screen
    Console.WriteLine("4 X 4 = " + oCalcProxy.Multiply(4,
4));
  }
}
```

Author Bio
Eilon Reshef is the vice president of Products at WebCollage. WebCollage provides comprehensive software solutions for interactive Web Services, allowing companies to package their existing customer-facing Web applications as Web Services and rapidly share them with multiple business partners.

EILON.RESHEF@WEBCOLLAGE.COM



Web Services What Do

Web services provide a method for encapsulating applications and making them usable in new contexts. Web services technologies are based on two concepts: a service-oriented para-digm and shared standards. The service-oriented paradigm supports one-time encapsulation of business processes and their usage in multiple locations. The second, a set of shared standards, allows businesses to expose, discover, and integrate these business processes. Ultimately, Web services enable new business opportunities by reducing business-to-business integration costs and allowing companies to offer their services through more business partners.

The ultimate adoption of Web services technology within organizations depends on the scalability it provides; that is, the degree of ease with which Web services can be delivered to business partners and incorporated into their business applications. The more effectively you can package applications as Web services, the more readily this technology model will be adopted across your organization and its business partners. So, broad adoption of Web services demands practical methodologies and straightforward technologies for sharing and integrating applications.

The potential of Web services has been mainly exploited in application-to-application programmatic interfaces. However, while this model lets application providers expose business logic to other applications, it doesn't address the Web user interfaces, that is, the way the functionality is offered to the end user.

In this article, I'll examine the structure of Web applications, and discuss the technical challenges that arise in integrating Web applications and, in particular, in building a Web user interface on top of Web services. I'll then demonstrate how Web user interfaces can be shared as part of Web services to make the services more accessible to business partners, and to reduce time-to-market.

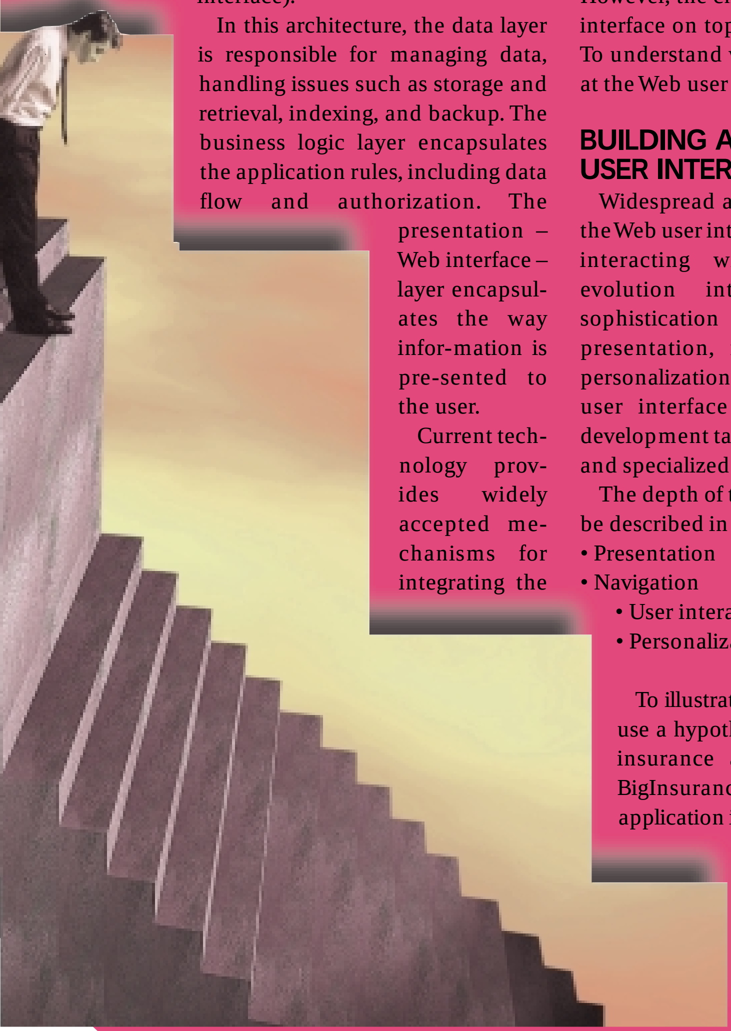
WEB APPLICATION ARCHITECTURE

To understand the requirements that enable application extensibility, we first need to observe how Web applications are built and integrated.

Enhancing the Web user interface
Enhancing the Web user interface

Enhancing the Web user interface

WRITTEN BY EILON RESHEF



Web services standards are closely linked to the conventional way of structuring applications as a set of three logical layers: data, business logic, and presentation (user interface).

In this architecture, the data layer is responsible for managing data, handling issues such as storage and retrieval, indexing, and backup. The business logic layer encapsulates the application rules, including data flow and authorization. The presentation – Web interface – layer encapsul-ates the way infor-mation is pre-sented to the user.

Current tech-nology prov-ides widely accepted me-chanisms for integrating the first two layers. XML offers a solution for integration at the data layer while programmatic Web services offer means for incorporating the business logic layer. However, the challenge of creating a Web user interface on top of Web Services still remains. To understand why this is so, we need to look at the Web user interface and how it works.

BUILDING A WEB USER INTERFACE

Widespread adoption of the Web has made the Web user interface the de facto standard for interacting with end-users. Its further evolution introduced new layers of sophistication and complexity that include presentation, navigation, interaction, and personalization. As a result, developing a Web user interface has evolved into a major development task that requires multiple teams and specialized skills.

The depth of today's Web user interfaces can be described in terms of four elements:

- Presentation
- Navigation
- User interaction
- Personalization

To illustrate how these elements work, I'll use a hypothetical example: an automotive insurance application provided by the BigInsurance company. A page of the application is shown in Figure 1.

Presentation is the way information is presented to the user. It includes visual elements, ranging from the application's style guidelines, through its concrete graphic design, to branding elements. These typically extend beyond the logo to the way products and services are offered to the user Presentation is central to application usability, providing convenient access to information, and it also includes the content presented to the user. In the BigInsurance example, the presentation of the application includes the brand at the top left side, and the color and font selection. The technologies typically associated with presentation include HTML, DHTML, JavaScript, and CSS at the client side, and XSL at the server side.

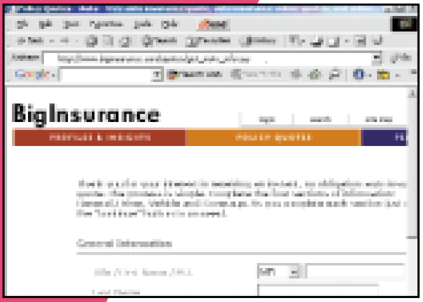


FIGURE 1 BigInsurance Web Application

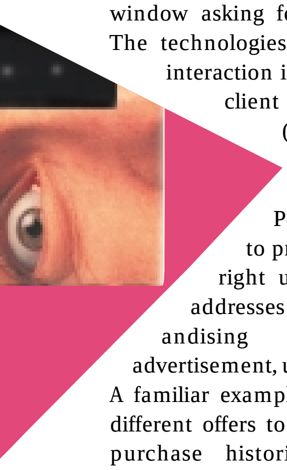
Navigation is the path users take to find relevant information within a more complex context. It embraces the overall application flow, from the links between applications to references to auxiliary information and global navigation bars. Navigation's important role in application usability is to ensure that users can find information. In the above example, navigation includes the global navigation bar at the top, as well as links such as the "Other Products" link at the top right corner. The technologies typically associated with

es it Take?

navigation include links, image maps, HTML forms and URLs – that is, making sure bookmarks work.

Interaction is the way the user interacts with the application, providing data and receiving information back. It includes many aspects of the application logic:

- Input workflow: what data is requested, and when
- Input validation: how the data is validated and how error messages are presented to the user
- Session management: supporting multi-page interaction



In the BigInsurance example, the interaction element determines the application flow. For example, after selecting the state and clicking on “OK,” the user is presented with a new window asking for additional information. The technologies typically associated with interaction include HTML forms at the client side, and dynamic pages (ASP/JSP) and state management at the server side.

Personalization is the ability to present the right offers to the right user at the right times. It addresses many e-commerce merch-andising concerns such as advertisement, up-selling, and cross-selling. A familiar example is the ability to present different offers to users based on their past purchase histories. But personalization extends beyond e-commerce. It can enhance Web funct-ionality by presenting different information views to users with different needs.

In the above example, BigInsurance might present different insurance products to the customer based on age, gender, or state. The technologies typically associated with personalization are:

- Session management and cookies at the technical level
- Template technologies such as ASP and JSP as the underlying mechanisms for custom-izing content
- Rule-based engines and collaborative filtering as concrete e-commerce personal-ization engines

Clearly, the development of the Web user interface represents a significant investment for a company in terms of both time and cost. Creating this interface requires the combined efforts of multidisciplinary teams that include business managers, information architects, graphic designers, HTML programmers, JSP programmers, and Java programmers. The development cycle is typically complex and lengthy.

Consequently, Web services providers must recognize that building a Web user interface on top of a Web service demands a sizable development commitment from each partner using the Web service. The magnitude of the effort required can itself defeat the whole point of Web services: to make it easy to work with many partners. Obviously, a methodology and a set of technologies that can eliminate this development requirement is needed to alter the equation in favor of creating and using Web services.

INTEGRATING THE WEB USER INTERFACE

One approach that doesn't require business partners to re-engineer the Web user interface is to package the user interface itself as part of the Web service. Sharing the user interface requires serving user interface elements – namely, HTML code – as part of the Web service. This can be implemented directly on top of standard technologies – SOAP, WSDL, and UDDI-relevant Web services can supply a programmatic operation that returns HTML elements to the Web service consumer.

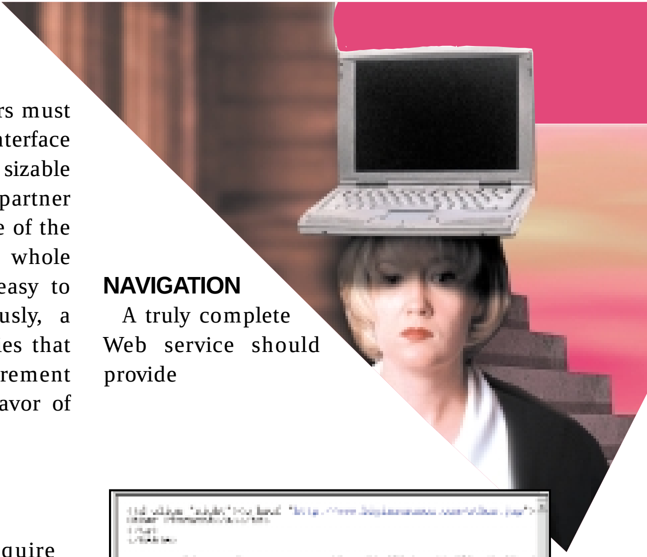
Figure 3 illustrates the concept. In this model, the Web interface layer is packaged as a part of the Web service into what I’ll call an “interactive Web service.” This Web service can be incorporated into other sites without redeveloping the Web interface. A thin layer that customizes the Web interfaces and integrates it into a new context may be used.

However, to share the complete Web user interface, you can’t share any HTML code. You must extend the same principle of shared functionality to all the facets of Web user interface: presentation, navigation, user interaction, personalization.

Handling those issues may require more planning and additional technologies. For example, the HTML code example in Figure 3 presents the code behind the a BigInsurance page, and illustrates the challenge of preserving a complete Web user interface across multiple sites.

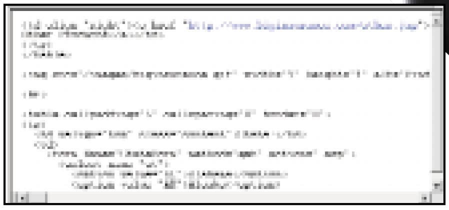
PRESENTATION

Because HTML wasn’t designed to be transportable, the code in the example above would not function properly when placed in a new location. You must at least “absolutize” the URLs of the images so they can be incorporated into an arbitrary location. In addition, you must resolve other issues: cascading style sheets (CSS), absolute positioning, JavaScript code, global document attributes – such as onLoad – and miscellaneous browser-specific HTML intricacies.



NAVIGATION

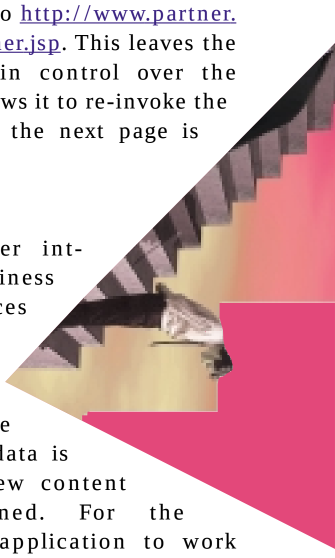
A truly complete Web service should provide



```
[illegible]
```

FIGURE 2 BigInsurance Web Application HTML

the application’s full functionality, not just a single page. For example, the HTML from Figure 2 contains an “Other Products” link that sends the user to another URL. Placing this HTML code in a new location would cause the user to navigate outside the context of the “container” site. To resolve these issues, you can dynamically “bend” the links; namely, change the links to point to a partner URL (e.g., to http://www.partner.com/next page=other.jsp. This leaves the host application in control over the interaction, yet allows it to re-invoke the Web service when the next page is required.



USER INTERACTION

Sharing the user int-eraction with business partners introduces yet another layer of complication. As the user int-eracts with the application, new data is submitted and new content must be returned. For the originator’s Web application to work correctly, the application must receive data from the user, yet the “host” appli-cation must be able to control the presentation of any information in its own context. In the code example, whenever the user submits the “State” form, the information is routed to the BigInsurance server while the resulting HTML is available to the partner using the Web service. You can accomplish this in several ways. For example, whenever the user submits data to your application, you can save the resulting HTML and redirect the user to the partner site. Whenever the partner re-invokes your Web service, you can return the saved HTML code to the partner application.

PERSONALIZATION

Sharing personalization elements with business partners isn't a straightforward problem. Typically, Web applications rely on temporary and permanent cookies to store user session information. Personalization logic is then based on that information. When HTML code is transported to a business partner server, user information is "lost" so the originator's application loses visibility of the user. Depending on the business requirements, you may need to require partners using your Web service to maintain cookies and URL parameters, and communicate them back to the Web service.

SUMMARY

Web services are the new standard for encapsulating business logic and sharing it with business partners. The ability to implement Web services is essential to keeping abreast of Internet evolution, and a successful Web services strategy relies on easy integration of Web services with business partners. Key to that ease of integration is the creation of a customer-facing Web user interface that exposes the underlying functionality to the user.

As part of their implementation of Web services, businesses must decide whether to require partners to build the Web user interface or to provide it as part of the Web service package. In many cases, encapsulating interactive elements within the services reduces implementation costs and increases their ability to work with more business partners.

Packaging the Web user interface with the Web service lets companies appreciably simplify Web services adoption by their business partners and create easily re-usable components. Service providers can leverage their existing investments in Web user interfaces. Their partners can utilize proven customer interfaces.

With relevant methodologies and technologies in place, companies can eliminate the re-engineering of the Web user interface, and create an innovative model that brings businesses significantly closer to fully realizing the potential of Web services. ⓔ

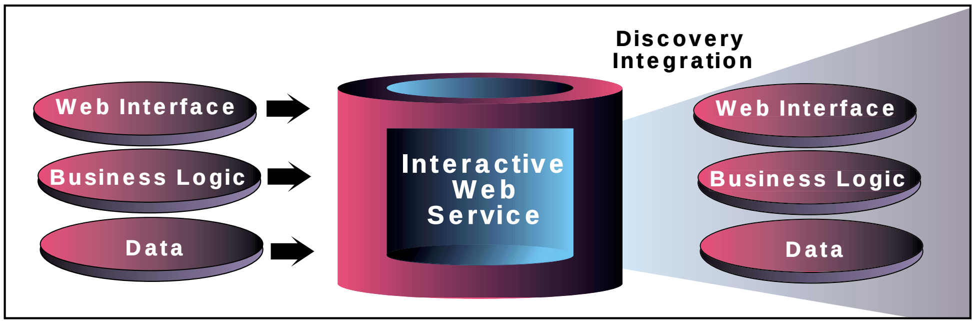


FIGURE 3 Encapsulating complete Web applications as Web services

BUSINESS & TECHNOLOGY wireless
Begin your unwired experience now
information for the unwired world
www.wbt2.com

Written by Kurt Cagle

Rethinking Web Services

Can Web services live up to the royal hype?

Part 1

I've been at this game for a while, a fact that has been hammered into my awareness with distressing frequency of late. I worked with Hollerith cards in college, running my programs through a machine with a distressing tendency to shred my carefully typed code into so much confetti if the deck was not perfectly aligned in the bin. I can remember a time when mentioning the object-oriented programming paradigm was a sure invitation to fisticuffs between its adherents and the old guard. In fact, SQL hadn't even been conceived when I went through college.

I guess that puts me into the neo-ancients – not so far back that I can remember when punch cards were considered hot technology, when programming involved switching cables between sockets, and programmers still had the moniker of computer scientists and wore white coats, but at the same time old enough to call most of the leading players in the tech sphere kids.

The reason I bring this up has to do with an old, old idea in comparatively new clothing: Web services. It sounds shiny and new, an idea that resonates in the imagination. Get the refrigerator to order new milk when the old gallon goes out of date. Schedule appointments with the family dentist by negotiating his schedule with yours via complex Web services protocols. Order that shiny new sports car that you want with the cherry-red paint job and the custom interior, and the orders go off to the factory and tell the robots there what to do. Sounds wonderful. Who wouldn't love it!

Web services are being marketed as the coolest technology since the PC was invented, completely revolutionizing everything and ushering in a brand-new technology boom. Microsoft is at the forefront of this marketing, understandably, but most of the big software vendors are now entering into the fray with their own versions of Web services – not wishing to be left out in the cold and give even the hint of a competitive edge to Redmond.

Yet for all of the sudden movement in the industry towards the new testament of Web services, I think there are a number of unresolved questions that make me wonder whether Web services are being both seen and marketed incorrectly, and whether we are advancing into the fray of battles over whose version of SOAP is to rule supreme and who is to become the ultimate victor in "servicing" the Web before it's really all that clear that the war is worth fighting in the first place.

As such, I'm going to sit here, my ancient bones staring at the juggernaut of a dozen multibillion-dollar industries, and ask some questions about the emperor's clothes and his choice of tailors. (What they don't tell you in the fairy tale is that the kid who pointed out that the king was naked was summarily imprisoned and was last seen as a rower on a slave galley.)

AND THE ANSWER IS...

So here goes. Contestant No. 1, "What exactly is a Web service?"

"I know! I know! It's a call made by a software program or hardware device to another software program or hardware device asking for a SOAP message."

Well, sort of. SOAP has an interesting history. It started out as an answer to XML-RPC, a quick-and-dirty schema that Dave Winer of FrontierLand developed to help implement a number of programs, including a compelling cooperative news setup that is rapidly becoming a de facto standard.

The idea there was pretty simple: in a program, you can encapsulate a request against a given program (device, service) in XML, and get the information you need from it or get it to run some function, regardless of where the server is located. Now, except for the XML part, this is certainly not a new idea. COM and CORBA both exist to do much the same thing – a process called marshalling, in which a request from a client is sent across some kind of programming barrier (a different application, a different machine on the ethernet network, etc.).

The problem is that both suffer from much the same thing. COM is a binary protocol that is incompatible with non-Windows systems without some heavy-duty translation software, a process that exacts its tool on performance. CORBA is designed to be more open, but it has problems crossing the divide in the other direction because the COM and CORBA specs are sufficiently different that they can't be cleanly mapped from one to the other.

With XML-RPC it's reasoned that, for relatively low volume transactions where the cost of instantiating a DOM isn't that significant, you could get around many of the translation issues by using a text/XML structure that con-tained enough infor-mation to identify the transaction and carry a payload.

In many respects this isn't all that radically different from

Author Bio

Kurt Cagle is an author and developer specializing in XML, XSLT, and Web technologies issues. He is the president of Cagle Communications, located in Olympia, Washington, and can be reached either via his Web site at www.kurtcagle.net or by e-mail at

KURT@KURTCAGLE.NET

the way a typical HTTP POST message is sent: the structures involved are flatter and less rich, but you're still requesting information from a device or server of some sort. Moreover, the typical Web client/server architecture essentially has a program (the browser) sending a request/response series of messages to the server to build a Web page. The user may set up the location of the server in a URL, but there's absolutely no reason that it couldn't be another computer system initiating the request.

However, the concept solved a major problem that Microsoft ran into with the Internet. DCOM proved largely to be a bust: you needed to set up both systems to run DCOM, it required some deep programming skills, and it was far more complicated to work with over the largely stateless Internet. SOAP thus came out (powered largely by Microsoft) as a way of providing COM information between separate processes transparently. Somewhere along the way the notion that XML should be used sparingly because of its size and instantiation costs seemed to get lost, and SOAP became the first strike that Microsoft would make into the world of an Internet COM.

THE TRANSPORT PROTOCOL OF CHOICE

SOAP itself has evolved somewhat (and watching the SOAP discussion board, I'm using the word *evolve* here in a very...um, tempered...way) and been taken up by the W3C as the de facto standard for the XML Protocol working group. It was also adopted in March as the transport protocol of choice for the ebXML e-business initiatives that OASIS is developing.

Some compromises were made. The original SOAP spec wasn't designed for passing non-XML payloads, which still comprise the bulk of all such material over the Web, but the current incarnation of SOAP making its way through the XML Protocol committee does address this issue ... somewhat.

The payload issue highlights one of the other characteristics of SOAP, as well as bringing up another question. SOAP is often described as an envelope, or more properly, a set of protocols for transferring envelopes. In an ideal world, a SOAP message should in fact know nothing about its payload beyond the critical information of who the payload is from, who it is addressed to, and who it should be sent back to; however, like other electronic messaging systems, things aren't always that simple.

What about the situation where a SOAP message is sent to multiple servers or is intended to be forwarded to other addresses? What, in fact, constitutes an address, anyway? What should the recipient do upon taking in the message? Should information about the fact that the recipient was unable to work with the package, even if it was valid, be sent back? What constitutes an error? What responsibility does the client have to ensure that the recipient can understand the message?

These are all questions that are currently hotly debated in the Web services world, and the fact that there are any number of companies jumping on the Web services bandwagon even before many of these questions are adequately resolved is somewhat worrisome.

There's a second, not-so-subtle concern here. A SOAP message is an envelope; envelopes have been opened, examined, and resealed before, sometimes with contents altered, other times with their contents examined. SOAP is no different, and in fact the text nature of the contents makes SOAP even more transparent to snooping than binary messages would be.

This in turn means that in many applications, either security will need to be applied over the network itself to prevent intrusion and interception (something done currently with SSL) or the security will have to be provided at the encryption level. A central tenet to such security is that you basically attempt to hide as much of the relevant information as possible – if the payload is encrypted but the header contains the methods to be called in plain-text, this acts as a flag to would-be hackers of the system that indicates what the content is about.

This isn't an insurmountable

problem, but it does lead to another one that is perhaps more pertinent. The more secure a message is, the more horsepower and time it takes to perform both encryption and decryption, and the larger the messages themselves become.

Couple this with the fact that SOAP messages are typically used in an RPC fashion – they are called autonomously by a software program operating at speeds several billions times faster than human beings, and the question naturally arises about how quickly the server will get overwhelmed.

DDOS ATTACK CAN INCAPACITATE

Before dismissing this complaint out of hand, keep in mind that a denial of service attack on a server is precisely this kind of a system – automated, asynchronous, anonymous – and it's instructive to realize that most of the companies that have been pushing Web services technology have also been incapacitated at one time or another by a DDOS attack.

Add to this another, fairly insidious facet of working over the HTTP protocol. HTTP is asynchronous. Asynchrony has a number of advantages in general over synchronous calls in normal HTTP usage. For instance, you can scale far more effectively with asynchronous calls and can increase the number of potential connections exponentially because they aren't really connections in the stateful sense.

Most RPC calls are made over stateful connections, however – and usually for good reason. It's easier to secure a stateful line. You can optimize stateful connections, and enveloping (and hence the sizes of those envelopes) can be made more efficient. You don't have to interrupt a stateful connection to process another message on another thread. It more easily models the behavior of stateful connections within local COM-type systems.

On the other hand, it also significantly cuts down on the number of clients you can offer your service to. This holds for RPCs in general, and raises some questions about the long-term wisdom of attempting to perform step-wise computations through a series of RPCs on a distant machine, which is in essence the COM model writ large.

On a single dedicated network that services a given organization, such control is not noticeably a bad thing, but over the slow, frequently unreliable Internet too extravagant a use of RPCs could potentially bring the entire network to a crawl (the tragedy of the commons, writ globally).

There is an alternate model, one that in fact works quite well and is not all that radically different from where Web services are developing now.

A DIFFERENT MINDSET

The central idea is to batch RPCs into a single process, transmit this set of instructions to the server asynchronously, have that computer perform the actions that it needs, then pass that information back. This is a goal that is also more consistent with XML in general, since XML generally works best when visualized as the encapsulation of a process rather than a single action. It requires a different mindset from the way that COM apps are programmed – those focus on a series of discrete actions and are ultimately synchronous, even if threaded.

Of course, the danger to many software vendors in particular is that such a methodology doesn't necessarily fit well with their existing offerings. It's sexier to market the ability to have a command or function call that can operate transparently (regardless of where the machine is) than it is to make developers think about the processes that they want to operate remotely. It's also easier to have those same programmers work poorly in their existing paradigm rather than work more efficiently in a different paradigm.

XML is a different way of thinking. A good XML developer can tell you that the approach you take to working with XML data varies considerably from the way you handle object-oriented code; that it is far more oriented toward the manipulation of sets of information rather than property-driven objects. This thinking extends to SOAP and Web services, which can in fact be powerful, but only if people stop trying to make SOAP into an object container.

PERSONAL SPACE

Okay, that's one simple question with a remarkably complex answer. The next question falls into the journalistic vein. Contestant No. 2, "Who needs Web services?"

"I know! Web services will replace modern software, will make it easier to do business-to-business transactions, and will make it possible for consumers to talk to their toasters!"

Now personally, if any of you have a compelling need to talk to your toasters, over SOAP or otherwise, then you probably have bigger problems than which Web protocols to use.

Seriously, Web services are often invoked for use in two fairly distinct areas. The first lies in the consumer market, where Web services will replace the Internet as we now know it, while the second is in the business-to-business sphere, where Web services will replace EDI as we know it.

POUNDS OF SALT

I think in many respects that, contrary to the analysts at any number of multibillion-dollar business companies, neither one of these are terribly well suited to Web services at all. Of course, the analysts at these conglomerates make six figures and in most years I'm doing well making five, so take what I have to say with a healthy pound or two of salt.

THE EXTREMELY EFFICIENT HOME

Look first at the consumer market. The vision here seems to be one of those concocted by the same people who have been pushing the completely automated home, one of extreme efficiency and interconnectedness where human beings don't need to worry about those messy interactions with other human beings. The scenario runs something like this: Janet is heading off to work when her PDA beeps at her indicating that her son, Tim, needs to get his cavity filled by the dentist. She presses a button and her PDA sends SOAP messages off to the dentist that negotiate a time when both her schedule and the dentist's are open. The message gets sent to Janet's calendar, which updates itself and sets up the necessary alarm.

No need to play phone tag, completely transparent, the ultimate plug-and-play operation. It neglects, however, the fact that you're not dealing with scenarios here; you're dealing with people.

Most people don't have their lives scheduled to such a degree that they can have an automated process do all of these negotiations for them. The interfaces involved, if current interface design is any indication, will probably end up taking far more steps than a simple phone call would. Most software, in fact, tends to limit the flexibility of design in favor of the cost of coding. In my experience taking myself and a school-age kid to the dentist many times, the appointments are typically made at the end of the previous appointment.

Finally, there is an implicit assumption on the part of the evangelists that Janet's calendar and the dentist's appointment system all agree on a common protocol, which means that they must subscribe to the same set of services. If Janet and the dentist happen, heaven forbid, to belong to different software services, does this mean that Janet can't go to this dentist?

I've heard a number of similar consumer-based scenarios, and what I find most interesting is that they all seem to share a number of common faults:

• ***Lack of Social Interactions***: In the personal sector, interactions serve many more purposes than the simple ones of exchanging information. In a non-networked computer world, this wasn't so much of an issue, because the majority of computerized interactions were largely related to the immediate task at hand.

As we move increasingly into a situation where computers could handle tasks that have up to now been largely social or communicative, there's going to be increasing resistance on the part of people who feel they're giving up social interaction for machine efficiency. (If there's any real doubt about this, consider the scan-it-yourself grocery stores, which have discovered that after the novelty has passed, most people stop using the scan lanes in favor of human interaction. There's a lesson here for any service provider.)

• ***Efficiency vs Flexibility:*** Computers are the ultimate efficiency machines. There are few tasks that couldn't be made more efficient; however, one of the first casualties of any system where efficiency is optimized is flexibility. XML by itself is not terribly efficient precisely because it is designed to be flexible, and a good XML design can often handle contingency cases in ways that more traditional code (and make no mistake about it, Web services, for all their XML underpinnings, are **very** traditional code) will choke on. Personal services are hard to create precisely because they are unpredictable, and any service that fails to handle the vagaries of human interaction will not be accepted by people.

•***Semantic Incompatibility:*** The question about Janet and her dentist being on different protocols is an example of semantic incompatibility. The vision that underlies Hailstorm (and [not to pick on Microsoft] is pervasive in many other Web services offerings as well) is that one company essentially becomes the broker for *every* API out there. Your calendar and my scheduler will work fine if both happen to use the same version of the calendar Web service API, but simply because both use Web services doesn't guarantee that they'll be able to talk to you. The solution according to Microsoft is simple: you have to move to a dentist who does support your service. Think about that one... carefully.

• ***Utility vs Privacy:*** Another common scenario in the direct personal sphere is the notion of being able to call up from your cell phone to notify the thermostat to turn itself on as you're coming home, to start thawing the chicken for supper and to start the e-mail download. Telematics, the industry of automating objects via external commands, sounds cool in theory, but in practice never lives up to the hype.

There may be a few people who love gadgets so much that they couldn't live without the poultry defroster and the thermostat control system, but in practice the efficiencies of time don't justify the worry. If the television could potentially record and periodically transmit what you watch, then it can also charge you for certain shows that used to be free – and also provide a powerful tool to advertisers about what does and doesn't motivate you. Remember, any pipe that comes in also has to go out.

SUMMARY

I guess, in the long term, I worry that the marginal benefits most people would personally get out of the use of Web services come at the potentially high cost of letting companies control even more of their lives than they do now. There are exceptions, which I discuss in Part 2 of this article (appearing next month), but the potential for abuse of Web services in the personal sphere is strong enough that people should seriously explore before letting it become a dominant feature in their lives.

Next month, in Part 2 of this article, Cagle covers B2B integration, intra-business applications, peer-to-peer, and the decentralization of services.

WEB MADE EASY

Written by Graham Glass

The first step towards collaborating software services

Author Bio

Graham Glass is the chief architect and founder of The Mind Electric. He believes that the evolution of the Internet will mirror that of a biological mind, and that architectures for helping people and businesses to network effectively will provide insight into those that wire together the human brain. His passion is creating the kind of world that currently only exists in science fiction books. Graham earned his BSc in Mathematics and Computer Science from the University of Southampton and his MS in Computer Science from the University of Texas at Dallas.

GRAHAM@THEMINDELECTRIC.COM

Web services are XML-based building blocks for assembling distributed systems. Now that it's everyone's favorite buzzword, many developers are interested in experimenting with the technology to see what the excitement is all about.

The first generation of Web services platforms, such as Apache SOAP, was great for getting services up and running, but their ease of use and performance left a lot to be desired. The second generation of platforms is intuitive, fast, and can dramatically reduce the learning curve associated with Web services.

SERVICES

This article describes one of these platforms, called GLUE, which is free for most commercial uses. I hope that it gives you a taste of how much fun it can be to work with Web services!

Before delving into a simple example, let's quickly review the technologies that make up a complete Web services platform.

THE WEB SERVICES TECHNOLOGY STACK

There are three standards that most Web services platforms implement:

• ***SOAP*** (Service Oriented Archit-ecture Protocol) is the standard for network communication between software services. SOAP messages are represented using XML and can be sent over any transport layer. HTTP is the most common transport layer, with implem-entations also available for SMTP, JMS, and the IBM MQseries.

• ***WSDL*** (Web Services Description Language) is the XML equivalent of a resume and describes what a Web service can do, where it resides, and how to invoke it.

• ***UDDI*** (Universal Description, Discovery, and Integration) is the standard that allows information about businesses and services to be electronically published and queried. UDDI is the Web services matchmaker and enables the creation of fluid, dynamically-assembled systems.

Implementations of these standards are normally hosted within an application server that provides a Web server, servlet engine, XML processor, and other supportive infrastructure. A complete Web services technology stack thus looks like Figure 1.

GLUE includes an implementation of this entire techn-ology stack in a 360K .jar file, which reduces setup problems and allows it to be used in almost any kind of application.

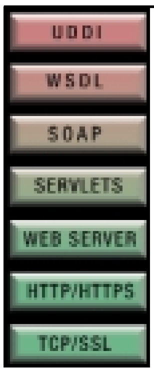


FIGURE 1 **Web Services Technology Stack**

GLUE can be used standalone or within a third-party servlet engine such as Tomcat or BEA WebLogic.

The examples discussed here were run in a standalone configuration.

BUILDING YOUR FIRST WEB SERVICE

The quickest way to get a feel for Web services is to build a few programs. This section will guide you through the creation of a simple Web service that you can run on your local computer or across the Internet. It's particularly fun if you can team up with a friend and invoke Web services between your home computers. A link to the source code for these examples is provided at the end of this article.

Table 1

Java type	XSD type
boolean	boolean
byte	byte
char	unsignedShort
short	short
int	int
long	long
float	float
double	double
String	String
Date	dateTime
BigDecimal	Decimal

The example in Listing 1 is a simple currency exchange service that returns the conversion rate between two countries. The Java interface for this service is straightforward, and defines methods for setting and getting the current value in US dollars of a particular country's currency as well as a method for obtaining the exchange rate between two country's currencies.

Most Web services platforms provide built-in support for all of the standard Java data types and some common utility classes, mapping them onto the XSD data types as shown in Table 1.

Byte arrays are converted into base64Binary, which is an efficient XSD encoding for binary data. All other single-dimensional arrays are mapped into their XSD equivalents. In addition, custom serializers allow objects to be sent between SOAP systems in a neutral and portable way.

Listing 2 is the implementation of Exchange, which stores currency values in a Hashtable.

GLUE has two key classes for building and deploying Web services. The first is **Registry**, which includes static methods for publishing Web services and binding to them. These methods are:

publish(String path, Object object): Publish the object to the specified path. By default, all of the object's public static and instance methods are exposed and can be invoked by SOAP clients.

bind(String path, Class type): Return a proxy to the service described by the specified path and that implements the specified interface type. The path can be the local name of the service or the URL of its WSDL file.

unpublish(String path): Unpublish the object from the specified path.

The second is **HTTP**, which provides static methods for starting up an in-process Web server that can accept incoming SOAP requests:

startup(String path): Start a Web server on the specified path.

shutdown(): Shut down any Web servers that were started using startup().

Listing 3 starts up a Web server on port 8004 of the local host to accept messages arriving on /soap. It then publishes an Exchange object as "exchange" for use by any SOAP client.

Note that the server does not exit when the last line is reached because the Web server has threads running, and Java does not halt until all threads have completed.

The client program binds to a Web service by invoking Registry.bind() with the path to its WSDL file. Because Java is a statically-typed language, the proxy that is returned by the binding operation must be cast to the interface of the appropriate type. Once the proxy is obtained, the client uses it to invoke the service as if it were a local Java object. A GLUE client can bind to any SOAP service using this approach (see Listing 4).

A GLUE server dynamically generates WSDL for the services that it hosts; if an incoming request for a file ends with .wsdl, it sees if the file name matches that of the path of a local service. If this is the case, the GLUE server examines the local Web service, generates and caches its WSDL description, and returns the WSDL file to the requestor.

To run the example, execute the ExchangeServer program in one window and the ExchangeClient program in another window. Here is the output for ExchangeServer:

```
> java examples.ExchangeServer
GLUE (c) 2001 The Mind Electric
startup server on
http://199.174.53.144:8004/soap
```

Here is the output for ExchangeClient:

```
> java examples.ExchangeClient
usa/japan exchange rate = 2.5
> _
```

If you want to run the same example across the Internet, simply replace the URL in the ExchangeClient with the URL of the remote server.

MANAGING SERVICES FROM A BROWSER

Once you've built your first Web service, it's easy to interact with it from a Web browser. You can enter the URL of the SOAP endpoint into the browser and see the services, as well as invoke them directly from the console.

Figure 2 is a screenshot of a GLUE console that shows the exchange rate service, including its address and a list of its methods.

You can use the console for invoking the methods from the browser, and for viewing other Web services anywhere on the Internet.

INVOKING REMOTE SERVICES

Web services are all about building applications from network building blocks, and one of the best sites for experimenting with third-party services is XMethods, located at www.xmethods.net.

FIGURE 2 GLUE console showing exchange rate service

When you visit XMethods, you'll see a listing of Web services on their home page. Each service has a brief overview, which includes the hosting company, the name of the service, a description, and the Web services platform that it's running on.

When you click on a particular service in their listing, you're shown a page that contains details of the service, including its description, owner, location and WSDL (see Figure 3). The WSDL is the most important item, since it contains everything you need to invoke the service.

For example, the delayed stock quotes service has a WSDL file located at http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl.

Most Web services platforms include a utility for automatically creating client-side access code from a WSDL file. For example, to generate a Java interface and helper class for

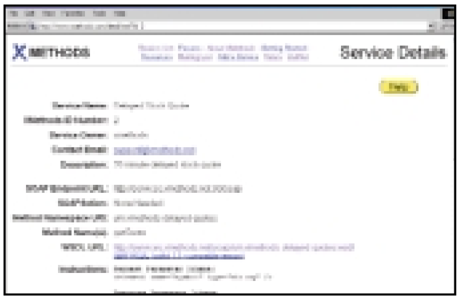


FIGURE 3 Page detailing the services

the XMethods Stock Quotes service, you can enter the following GLUE command, using the –p option to place the code into the examples package.

```
> wsdl2java
http://services.xmethods.net/soap/urn:xm
ethods-delayed-quotes.wsdl -p examples
write file IStockQuote.java
write file StockQuote.java
>
```

The generated IStockQuote.java interface is a regular Java interface with no coupling to anything that is Web service specific. The name and argument types of each method are extracted automatically from the WSDL file.

```
package examples;

public interface IStockQuote
  {
  float getQuote( String symbol );
  }
```

The StockQuote.java helper class simplifies access to the remote Web service. It defines a couple of static methods: one that returns a proxy to the specific Web service, and another that returns a proxy to a similar service but at a different specified location.

```
package examples;

import electric.registry.Registry;
importelectric.registry.RegistryException;

public class StockQuote
  {
  public static IStockQuote bind() throws
RegistryException
    {
    return bind( "http://services.xmeth-
ods.net/soap/urn:xmethods-delayed-
quotes.wsdl" );
    }

  public static IStockQuote bind( String
url ) throws RegistryException
    {
      return (IStockQuote) Registry.bind(
  url, IStockQuote.class );
      }
    }
```

To access the XMethods Stock Quotes service from an application, call StockQuote.bind(). The following client gets an IBM stock quote using this approach.

```
  package examples;

  public class QuoteClient
  {
  public static void main( String[] args )
    throws Exception
    {
    IStockQuote quotes =
StockQuote.bind();
     float ibm = quotes.getQuote( "IBM"
);
     System.out.println( "IBM quote is "
+ ibm );
     }
  }
```

To run the example, execute the QuoteClient program from a window. Here's the output you should see:

```
> java examples.QuoteClient
IBM quote is 116.1
> _
```

Once you get used to building simple services, the next step is to build services that orchestrate groups of other Web services. Unfortunately, service aggregation is beyond the scope of this article.

EPILOGUE

Although this article only scratched the surface of Web services, I hope it encourages you to take the next step and start building Web services of your own.

I think that Web services are the most exciting thing to happen in the world of computing for a long time, and the dream of a smart network teeming with collaborating software services is getting very close.

RESOURCES

GLUE home page: http://www.themindelectric.com/products/glue/glue.html

XMethods: http://www.xmethods.net

Source code of examples: http://www.themindelectric.com/glue/syscon.html

Listing 1

```
package examples;

/**
 * An interface for getting exchange rates.
 */
public interface IExchange
  {
  /**
   * Set the value, in US dollars, of the specified
currency.
   * @param country The country.
   * @param value The value in US dollars.
   */
  void setValue( String country, double value );

  /**
   * Return the value of the specified currency.
   * @param country The country.
   * @return The value in US dollars.
   * @throws ExchangeException If the country is not
recognized.
   */
  double getValue( String country ) throws
ExchangeException;

  /**
   * Return the exchange rate between two countries.
   * @param country1 The country to convert from.
   * @param country2 The country to convert to.
   * @return The exchange rate.
   * @throws ExchangeException If either country is not
recognized.
   */
  double getRate( String country1, String country2 )
    throws ExchangeException;
  }
```

Listing 2

```
package examples;

import java.util.Hashtable;

/**
 * Simple implementation of IExchange.
 */
public class Exchange implements IExchange
  {
  Hashtable values = new Hashtable(); // in US dollars

  public void setValue( String country, double value )
    {
    values.put( country, new Double( value ) );
    }

  public double getValue( String country )
    throws ExchangeException
    {
    Double value = (Double) values.get( country );

    if( value == null )
      throw new ExchangeException( "country " + country + "
not recognized" );
    return value.doubleValue();
    }

  public double getRate( String country1, String country2 )
    throws ExchangeException
    {
    return getValue( country1 ) / getValue( country2 );
    }
  }
```

Listing 3

```
package examples;

import electric.service.registry.Registry;
import electric.server.http.HTTP;

public class ExchangeServer
  {
  public static void main( String[] args )
    throws Exception
    {
    // start a web server on port 8004, accept messages via
/soap
    HTTP.startup( "http://localhost:8004/soap" );

    // initialize an instance of Exchange
    Exchange exchange = new Exchange();
    exchange.setValue( "usa", 1 );
    exchange.setValue( "japan", 0.4 );

    // publish an instance of Exchange
    Registry.publish( "exchange", exchange );
    }
  }
```

Listing 4

```
package examples;

import electric.registry.Registry;

public class ExchangeClient
  {
  public static void main( String[] args )
    throws Exception
    {
    // bind to web service whose WSDL is at the specified
URL
    String url =
"http://localhost:8004/soap/exchange.wsdl";
    IExchange exchange = (IExchange) Registry.bind( url,
IExchange.class );

    // invoke the web service as if it was a local java
object
    double rate = exchange.getRate( "usa", "japan" );
    System.out.println( "usa/japan exchange rate = " + rate
);
    }
  }
```

Written by Clay Shirky

What Can Web Services Learn from P2P?

Becoming a more flexible, and more valuable, technology

Both peer-to-peer (P2P) and Web services are attempts at decentralizing computing. P2P is more mind-set than technology, an attempt to weave the world's intermittently connected machines, into the fabric of the Internet. Web services are a more formal technological challenge, an attempt to apply the Web model for publishing – loosely coupled components with a simple request-and-response model – to computing, using XML in place of HTML.

It's easy to see how Web services are affecting P2P, as many P2P companies are adopting Web services standards, such as defining their document formats in XML or exploring the use of UDDI as a P2P Registry. But what can Web services learn from P2P? The development of P2P applications has three lessons for Web services developers.

HTTP ISN'T THE ONLY TRANSPORT

Much of the Web's success can be traced to HTTP, which struck a careful balance in its synchronization between client and server. HTTP operates in real time but is stateless, which allows for tight coordination between a browser and a server, with very little overhead.

This trade-off, though, isn't the correct balance for all possible means of communicating. In particlar, HTTP is inadequate at either extreme of synchronization requirements. P2P Instant Messaging systems like ICQ and Jabber need their own protocols to manage the kind of flexible two-way stream required for instant messages and chat. At the other end of the synchronization scale, where requests to remote machines need to be batched or will take longer than a couple of minutes to execute (such as complex database interactions or requests that require significant computational modeling on the server), HTTP is also inadequate. Many P2P systems, like 3Path and InstantPowers, have adopted SMTP, the e-mail delivery protocol, to send asynchronous messages between nodes.

P2P demonstrates the range of services that can be built on top of alternate protocols, and since applications like Jabber and 3Path can handle XML natively, they're ideal for adoption (and adaptation) for Web services that need either tighter or loooser integration between client and server than HTTP allows.

DNS IS NOT THE ONLY ADDRESSING SCHEME

A central problem for P2P apps is how to address resources on PCs that have no permanent IP address and are therefore outside the DNS system. The typical P2P solution has involved a directory of user-created addresses, continually remapped to the user's current IP address, as with Napster or Groove. These systems have evolved into something more than just a dynamic replacement for DNS, though. In many cases P2P addresses are for resources other than the machines themselves. A Jabber address (or indeed any IM address) points to a person, no matter what machine they are using at the time, while systems like Freenet and MojoNation point to addresses for pieces of content, irrespective of what machines it is on.

For Web services, alternate addressing schemes mean that a Web service could be accessed at a Jabber ID rather than a domain name, and could listen for remote calls packaged as instant messages. Likewise, a Web services client that knew how to interoperate with P2P file-sharing networks could extract a piece of content by its address without knowing where that content lived.

DON'T DIVIDE THE WORLD INTO CLIENT AND SERVER

On the Web the roles of client and server are largely fixed – Yahoo is always the server, your browser is always the client. In P2P systems, those roles are temporary, with the same machine performing some of the functions of a client and some of the functions of a server, often at the same time. In the Gnutella network, for example, PCs act as servers for remotely requested files and as routers for other users' requests. This lack of permanent distinction is reflected in the naming schemes of P2P systems, where the software running on the user's computer isn't called a client but a "clerver" or a "node" or a "tranceiver."

Because Web services clients and servers can both send and receive structured XML, clients can act as servers, and vice versa. P2P has demonstrated the immense value that can be created when groups of machines, act as both client and server, and Web services that adopt the same attitude can create new functions in which:

- Jobs are shared across multiple clients.
- Central databases, typically thought of as servers, can initiate requests to remote PCs.
- Clients message one another directly, without needing any central server at all.

Though Web services attempt to broaden the success of the Web into the domain of networked applications and business processes, the success of P2P demonstrates ways in which their broad goals – automatic communications using structured documents – can be supported, without conforming to the narrower technological dictates of the Webs.

Web services can operate with more or less synchrony than the Web by passing XML documents over other protocols like Jabber or SMTP. They can access a greater range or resources more directly by using non-DNS addressing schemes for entities like people and files. And Web services can provide a greater range of behavior between nodes by loosening the distinction between client and server. By adopting the work P2P systems have done in these areas, Web services can become a more flexible and ultimately more valuable technology.

Author Bio
Clay Shirky is a senior analyst with O'Reilly and Associates where he concentrates on decentralizing technologies, particularly Web services and Peer-to-Peer.
CLAY@SHIRKY.COM

web services EDGE conference&expo™

October 24–25, 2001

XML EDGE conference&expo 2001

October 22–25, 2001™

Santa Clara Convention Center Santa Clara, CA

FREE Registration

Register for one...attend both

See them First at XMLEdge2001

Fundamentally Improving the Speed, Cost & Flexibility of Business Applications

Many New XML Products Will Hit the Market in Q3 of 2001

Opening Keynoter

Charles Goldfarb

The Father of XML

SYS-CON MEDIA

Owned & produced by SYS-CON EVENTS

WEB SERVICES CEO PANEL

Litwack	**O'Connor**	**Kutay**	**O'Toole**	**Slama**	**Morris**
CEO	President	CEO	Chairman	CEO	CEO
SilverStream	Sonic Software	AltoWeb	Cape Clear	Shinka Technologies	IONA Technologies

www.SYS-CON.com

Platinum Sponsor: Gold Sponsor: ADOS Silver Sponsor: SilverStream Corporate Sponsors: Association Sponsors: OASIS XML

Media Sponsors: XML JOURNAL JAVA WebServices WebLogic WebSphere COLDFUSION wireless SD Times Programmer's Paradise

Cape Clear Software Adds iPlanet Support

(Walnut Creek, CA) – Cape Clear™ Software has announced that CapeConnect, its Web Services platform, has full built-in support for the iPlanet Application Server..CapeConnect also includes support for CapeStudio, a rapid application development tool for Web services, as well as updated interoperability with other SOAP-enabled products and an enhanced SOAP client builder. CapeConnect Two for J2EE with support for iPlanet is available immediately and can be downloaded from www.capeclear.com/download.

Sonic Software to Extend Its Web Services Offering

(San Francisco, CA) – Sonic Software is expected to announce it's entering the Web services marketplace at the upcoming Web Services Edge 2001 Conference in Santa Clara, CA, October 22–26. Expanding on their flagship product, SonicMQ, the newest offering will embrace the latest in open standards, such as UDDI, WSDL, and SOAP, allowing users to take full advantage of Web services interoperability.

New Web services features, combined with SonicMQ's patent-pending Dynamic Routing Architecture (DRA), will allow organizations to create highly distributed networks of interoperable applications while still relying on the proven strength of JMS. Developers will also be able to administratively choreograph the interaction between distinct application components, or services critical to e-business.

SonicMQ is the first J2EE-certified Java Message Service (JMS) MOM architected entirely on Java technology.

For the latest information and updates go to www.sonicsoftware.com.

Killdara Announces Vitiris Web Services Platform

(Almonte, Ont.) – Killdara Corp. has introduced the Vitiris Web Services Platform, designed for OCM and embedded use. The new product features a small footprint and is standalone, low-cost, and portable. Vitiris evolved from Killdara's Paraphrase Engine product, updated to support SOAP, UDDI, and WSDL standards. Vitiris Standard Edition is a low-cost, entry-level platform, requiring only on the J2ME. The Enterprise Edition adds support for JDBC database connectivity, PKI security, and advanced transport such as SMTP and FTP. www.killdara.com

BEA Ships Web Services Release of WebLogic Server 6.1

(San Jose, CA) – BEA Systems, Inc., has announced the availability of BEA WebLogic Server 6.1, the Web services release of its Java application server. Version 6.1 is the industry's first and only enterprise-class application server to provide automatic code generation of new and existing Java applications into Web services and offers new standards-based application integration capabilities and an advanced Java 2 platform.

The new release supports SOAP, WSDL, and UDDI. It enables developers to automatically deploy EJBs as Web services with no additional programming.

BEA WebLogic Server 6.1 is available for download at www.bea.com.

SilverStream Announces App Server with Full Web Services Capabilities; jBroker Web

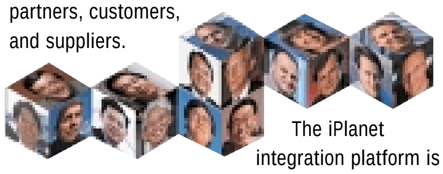
(Billerica, MA) – SilverStream Software, Inc. is shipping the SilverStream Application Server 3.7.3 with full Web services capabilities, including comprehensive support for XML, SOAP, WSDL, and UDDI. This application server features the SilverStream eXtend Workbench to streamline J2EE and Web services-based application development, as well as a high-performance Web services engine, jBroker® Web.

SilverStream Application Server is the foundation for SilverStream eXtend and provides users with an open platform for efficiently building, deploying, and managing mission-critical e-business applications. SilverStream eXtend allows organizations to exploit the value of their existing systems and rapidly deliver business applications to any user on any device or platform. SilverStream eXtend Workbench provides tools and wizards designed to simplify and accelerate the development of J2EE and Web services–based applications. jBroker Web is the core Web services implementation for SilverStream eXtend and also deploys to J2EE application servers.

The Web services release of the SilverStream Application Server with jBroker Web and the SilverStream eXtend Workbench is available immediately. www.silverstream.com

iPlanet Launches Most Complete Web Service–Enabled Integration Platform

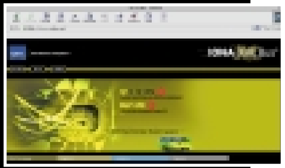
(Santa Clara, CA) – iPlanet E-Commerce Solutions, a Sun-Netscape Alliance, has announced the industry's first integration platform. The new platform incorporates SOAP, Java, and XML into one solution to allow customers to seamlessly link business processes and data to partners, customers, and suppliers.



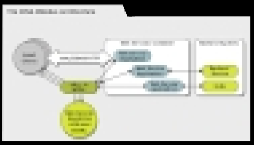
The iPlanet integration platform is part of the Sun ONE offering and includes iPlanet Integration Server, in both the B2B and EAI editions as well as iPlanet Message Queue for Java. www.iplanet.com, www.sun.com

IONA Releases XMLBus 1.2

(Waltham, MA) – IONA Technologies has announced that Web services created in .NET can be utilized in J2EE environments, and vice versa, using the IONA XMLBus Technology Preview Version 1.2. The new platform is designed to insure interoperability among .NET, MS SOAP Toolkit, and J2EE environments. It allows developers to build Web services from J2EE applications running on the IONA iPortal, BEA WebLogic, or IBM WebSphere application servers.



Other key features of the IONA XMLBus include support for SOAP with Attachments, allowing the development of Web services based on document passing style; and SSL integration to enable Web services to communicate via HTTPS; and JMX instrumentation to ensure that customers can administer and manage Web services in the same way that they manage their existing systems.



Developers can download a demonstration of this product at www.xmlbus.com.

SPECIAL INTRODUCTORY OFFER

SAVE $31*

HURRY, DON'T DELAY! OFFER EXPIRES: DECEMBER 31, 2001

Premiering... this *fall*

subscribe Now!

FOR FAST DELIVERY

The World's Leading Independent WebLogic Developer Resource

Helping you enable inter-company collaboration on a global scale

- Product Reviews
- Case Studies
- Tips, Tricks and more!

Go Online and Subscribe Today!

WebLogicDevelopersJournal.com

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebLogic.
*Only $149 for 1 year (12 issues) regular price $180.

SYS-CON MEDIA

The White Knight Killer App

WRITTEN BY

Steve Benfield

Steve Benfield is chief technology officer for SilverStream Software. SilverStream eXtend is the first complete environment for delivering service-oriented applications. SilverStream eXtend enables businesses to create, assemble, consume, and deploy Web services through J2EE or pervasive legacy integration.

Steve can be contacted at sbenfield@silverstream.com

Hmm..according to my latest Monthly Hype magazine, Web services open a whole new vista of applications to the world. Applications that have been unseen or unimagined by mankind. Everyone is asking about the Web Services Killer App. What is the Killer App?

The Web Services Killer App was unseen or unimagined by mankind until Tuesday, February 6, 2001, at 3:42 a.m., when I dreamed that I saw a great light. I walked to the light and I clearly saw SOAP and WSDL and UDDI and a few fuzzy things like security and transaction handling all dancing in a mist.

Suddenly the mist cleared and I saw a multitude of systems working together as if it were all part of a seamless machine. Then I awoke and realized I knew what the Killer App was.

What is the Killer App for Web services? What makes them pervasive? What will make a lot of money for a lot of vendors and consultants?

First things first.

The Killer App is integration.

"What?" the shocked audience gasps. "Hey, we want our money back. Integration? What do you mean? That's what Web services is about, but what's the Killer App? It can't be just plain old integration, can it?"

My position is that, yes, integration is the Killer App for Web services. It's that simple. I know it's not as exciting as Lotus 1-2-3 or e-mail or Doom, but that's what it is.

Big systems talking to other big systems – within and across business boundaries.

I know what you're saying, "But Steve, I could have guessed that!" Yes, you could have, but I jumped the line and told you before you could tell me.

The Killer App is the ability to actually build and integrate our software without fear.

Sending XML over HTTP is a pretty simple concept. Many companies are doing that today and Web services adds a standards layer, so we're no longer one-offing the effort. It's so simple in concept compared to other technologies that it's easy to dismiss its full impact.

Many times we see examples of technology looking for a problem. In the case of Web services, it's a pervasive problem latching onto a technology. The pervasive problem is integration.

Web services standards need to mature and to continue to grow—and they will. However, for day-in-day-out developers and IT managers who are trying to satisfy the needs of the businesses they serve, Web services deliver on a promise: the promise that you can separate technical implementation from business intent.

Every company on the planet needs to be rescued from its integration problems

Normally, when someone builds an integration these days it tends to be specific. It's either specific to a particular integration project – I need to hook System A to System B – or it's an attempt to create a generalized XML integration interface that the rest of the world can use. But it's not based on standard, so it's specific to a given custom protocol. Clearly, one-off integration projects are time-consuming and costly.

Every company on the planet suffers from integration problems. We'd all love to have a clean enterprise architecture. But we don't. Mergers. Acquisitions. Previous CIOs. Systems that were sold because the CEO's nephew was the salesperson. Divisions purchasing their own solutions to circumvent IT. Changing technologies. These have left us with a hodgepodge of different technologies in our organizations – and no one wants to pay to have these systems rewritten. Especially in today's economy.

So being able to integrate our systems internally adds value. Big time. And being able to create business processes across business boundaries adds even more value. It's easier to do ROI and cost-justification on the internal integration today because going outside the business means thinking outside the box – and having business partners who are like-minded. It will come, but it will take some time to get everyone on the same page.

So what is the Killer App?

It's having your Java application server be a Web services front-end for your mainframe and EDI transactions. Having a portal or JSP applications be able to use Web services calls to those transactions to create thin-client applications with straight-through processing to the backend. And then writing a Visual Basic application that calls those same Web services to create a client application, so your customer service people have an application with a rich GUI. And six months from now, allowing wireless devices to call those same Web services. And a year from now, having your business partners look up your services in a UDDI directory and call them. This is the kind of integration I'm talking about when I say it's the Killer App.

Being able to use the tools, languages, frameworks, app servers, data formats, and platforms of your choice without fear of massive integration issues is pretty killer to me – and to IT organizations worldwide.

SilverStream

www.silverstream.com