BETTING YOUR FUTURE: .NET OR J2EE?

December 2001 Volume1 Issue 3

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM

The Week That Was!
Web Services Goes Mainstream!
written by Robert McGarvey
PAGE 28

RETAILERS PLEASE DISPLAY UNTIL JANUARY 31, 2002
$6.99US $7.99CAN

SYS-CON MEDIA

BEA

www.developer.bea.com

BEA

www.developer.bea.com

SpiritSoft

www.spiritsoft.net/downloads

WebServices JOURNAL .NET J2EE XML

INTERNATIONAL ADVISORY BOARD
JEREMY ALLAIRE, ANDREW ASTOR, STEVE BENFIELD, DAVE CHAPPELL, BOB CROWLEY, GRAHAM GLASS, TYLER JEWELL, PAUL LIPTON, DAVID LITWACK, NORBERT MIKULA, BOB SUTOR

TECHNICAL ADVISORY BOARD
DAVE HOWARD, JP MORGENTHAL, DAVID RUSSELL, AJIT SAGAR, SIMEON SIMEONOV, RICHARD SOLEY, JAMES TAUBER

EDITOR-IN-CHIEF: SEAN RHODY
EDITORIAL DIRECTOR: JEREMY GEELAN
INDUSTRY EDITOR: NORBERT MIKULA
PRODUCT REVIEW EDITOR: JOE MITCHKO
.NET EDITOR: DAVE RADER
EXECUTIVE EDITOR: GAIL SCHULTZ
MANAGING EDITOR: CHERYL VAN SISE
SENIOR EDITOR: M'LOU PINKHAM
EDITOR: NANCY VALENTINE
ASSOCIATE EDITORS: JAMIE MATUSOW
BRENDA GREENE
ASSISTANT EDITOR: LIN GOETZ

WRITERS IN THIS ISSUE
ALAN CONWAY, COLIN EVANS, MIKE FARRELL, RAYMOND GAO, DAVE HOWARD, KARSTEN JANUSZEWSKI, ANDY LONGSHAW, BORIS LUBLINSKY, ROBERT MCGARVEY, NORBERT MIKULA, JOE MITCHKO, SEAN RHODY, MIKE RICHARDSON, HITESH SETH, CHARLES STACK, HAL STERN

PUBLISHER, PRESIDENT AND CEO: FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION & DESIGN: JIM MORGAN
SENIOR VP, SALES & MARKETING: CARMEN GONZALEZ
VP, SALES & MARKETING: MILES SILVERMAN
VP, EVENTS: CATHY WALTERS
VP, BUSINESS DEVELOPMENT: GRISHA DAVIDA
CHIEF FINANCIAL OFFICER: BRUCE KANNER
ASSISTANT CONTROLLER: JUDITH CALNAN
ACCOUNTS PAYABLE: JOAN LAROSE
ACCOUNTS RECEIVABLE: JAN BRAIDECH
ACCOUNTING CLERK: PATTI DEL VECCHIO
ADVERTISING ACCOUNT MANAGERS: MEGAN RING
ROBYN FORMA
ASSOCIATE SALES MANAGERS: CARRIE GEBERT
ALISA CATALANO
KRISTIN KUHNLE
CONFERENCE MANAGER: MICHAEL LYNCH
SALES EXECUTIVES, EXHIBITS: MICHAEL PESICK
RICHARD ANDERSON
SHOW ASSISTANT: NIKI PANAGOPOULOS
REGISTRATION ASSISTANT: JACLYN REDMOND
ART DIRECTOR: ALEX BOTERO
ASSOCIATE ART DIRECTORS: CATHRYN BURAK
LOUIS CUFFARI
ASSISTANT ART DIRECTORS: RICHARD SILVERBERG
AARATHI VENKATARAMAN
WEBMASTER: ROBERT DIAMOND
WEB DESIGNERS: STEPHEN KILMURRAY
CHRISTOPHER CROCE
JDJSTORE.COM: ANTHONY D. SPITZER

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT.
SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM
COVER PRICE: $6.99/ISSUE
DOMESTIC: $69.99/YR (12 ISSUES)
CANADA/MEXICO: $99.99/YR OVERSEAS: $129.99/YR
(U.S. BANKS OR MONEY ORDERS)
BACK ISSUES: $10 EA., INTERNATIONAL $15 EA
SUBSCRIBE@SYS-CON.COM

EDITORIAL OFFICES
SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9637
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

Shine a Little Light

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'm showing my age, but a number of years ago ELO released an album entitled "Discovery." One of the songs was entitled "Shine a Little Light," which is apropos for this month's editorial since our feature focus for this edition is *Discovery*.

I took part in a panel discussion on Web services during our Web Services Edge East show in New York City at the end of September. A number of very august industry representatives joined me, including James Gosling, Rick Ross, and David Litwack. The panel covered a number of topics that are on the minds of anyone considering Web services, and one of the topics that received a great deal of attention was Discovery.

Discovery is probably one of the most important differentiators for Web services as it positions itself as more than just the next incarnation of an RPC mechanism. It is the ability to locate a service without prior communication between service users that markedly advances the entire concept of assembly of systems from Web service components.

At least, it does in theory. The panel was skeptical on a number of issues. Our current idea of discovery mechanisms consists of UDDI and WSDL – two technologies that have a bit of overlap but more or less work together. UDDI is XML-based and as such is textually oriented, massively redundant in its meta-information, and extremely bandwidth intense. One of the issues we debated on the panel was the usefulness of a global UDDI. As we discussed the purposes and probable first adoption scenarios, it became clear that most of the industry expects Web services to be adopted first within the enterprise, rather than between enterprises. Not that we don't expect it to happen, but given the flurry of development that's gone on over the past few years, we saw industry trying to catch its collective breath by first consolidating their own internal systems.

So the usefulness, and even the implementability, of a global UDDI was brought into question. There was significant debate over the way UDDI would play. Most of us, myself included, thought that UDDI would first be deployed on an industry-by-industry basis, perhaps by third-party brokers in a fashion similar to the Net Market craze, or perhaps by key industry players in a preemptive strike. In doing so, the size of the overall database, and the network bandwidth required to support it, can be more thoroughly controlled.

Not everyone agreed though. Some thought that Web services should be like HTML, where anyone who wanted to expose something neat, something new, should be able to do so easily. Napster comes to mind as an example.

And speaking of Napster, the peer-to-peer model was discussed as well. James Gosling hit it on the head when he pointed out that peer-to-peer is just another, broader method of discovery. Once you identify – discover – the music you want, it's still a machine-to-machine connection. Which holds up well with my opinion that peer-to-peer is just another Web services option, not something completely distinct from Web services (although it can be).

We also discussed the versioning of Web services. The question of what to do when you've discovered a service and the owner changes it was on many people's minds. That actually led to a proposition about service-level agreements as an integral part of the UDDI entry, pointing out yet again how important discovery will be. For example, if the entry guarantees support for the service for one year, once you've found it via discovery you wouldn't need to look it up again, you could just use it. You would also be confident that the service interface would be stable for a period of a year, so that even if the owner of the service introduced an improved interface, you would have that period of time in which to investigate the improvement and adapt to it. In theory, a service is well defined and only the underlying implementation will change, but we all know that reality is never that simple.

Hopefully, this shed a little light on the topic of discovery. We have several articles this month intended to put it into the spotlight. Enjoy!

.NET MYSERVICES: A PERFECT EXAMPLE

It's worth mentioning that Microsoft's .NET platform does not specifically support most of the things I'm talking about in this article, but its upcoming .NET MyServices release is a set of Helper services that help .NET to cover some of these points. User identification, bill payment, and many other features are presented in this suite of Web services that will effectively extend the capabilities of the .NET platform (and presumably any Web services–enabled software) without the need for heavy infrastructure at every deployment site.

Author Bio:
Dave Howard was recently an architect with ObjectSpace, Inc., and a founding member of the OpenBusiness™ team, one of the first Web services platforms on the market. He is now freelancing in Dallas, Texas.

DHOWARD@THENETWORKEFFECT.NET

written by Dave Howard

Leveraging Web Services

Where are the Web Services that are going to change the face of computing?

What do you, the potential Web service provider, really want to get out of having published a Web service? A pat on the back? Notoriety? Fame and fortune? Garnering these elusive prizes from the available tools and platforms on the market today is difficult. But how about just a steady revenue stream and a loyal customer base? Let's take a look at how you can steer clear of the current roadblocks, so down the road you can implement a successful Web service offering.

Yielding to the Pedestrian

Have you looked for a Web service to use lately? If you have, chances are you've visited one of the public UDDI registries or maybe a Web services portal, such as xmethods.com or SalCentral. And if you've visited any of these places expecting to be blown away by the novel, innovative or just plain useful Web services they index, then you've probably been disappointed.

What you found was most likely a bunch of temperature and currency converters, simple addition widgets, and maybe a flight tracker. Where are the Web services that will change the face of computing, rewrite the history of the Internet, or at least live up to the Web services hype? Sorry folks, it's not there yet -- which is not a knock on the aforementioned Web service indexes. It's simply an observation that the maturity level of the publicly available Web service offerings is, well, premature. At the present, there are only a few that are truly helpful, but you have to look carefully to find them.

So, what gives? Why are the "Web services" out there mostly toys and experiments? For one thing, the standards involved are only beginning to firm up.

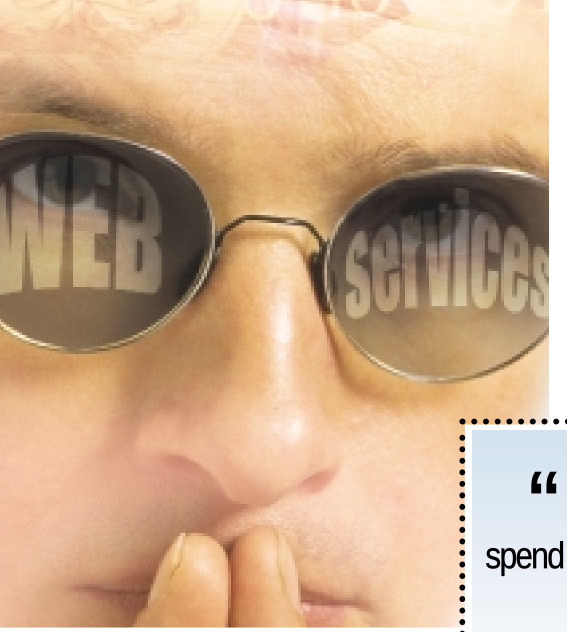
Several interoperability issues still surface when it comes to SOAP implementations, and the same is true of WSDL. And both the UDDI specification and implementations have a long way to go before they really make life easier for Web service providers and users.

In addition, the technology has only really been in front of developers for a few months and people have only just begun to think about what kind of services might actually catch on for users. Most of the Web services you find available now have the look of first attempts at getting "something" working…they're not even intended to be serious services.

Both of these explanations for the lackluster offerings are fairly obvious to anyone who has been scoping around Web services. But the real deep-seated block that keeps developers and strategists from taking Web services planning to the next level is beginning to reveal itself.

The Real Culprit

There are serious shortcomings in today's first-generation Web services platforms. In fact, it's difficult to justify a Web service. No one seems to know how it might be beneficial to the provider or how it might be useful to the consumer.



Why Are We Here?

What kind of a weighty question is that? Especially in a light technical piece. What I really want to know is, "Why are we talking about Web services in the first place?" Why are so many people interested in finding out how to publish Web services and make them available to users?

Tell Me What's Happening

According to the buzz you hear in conversation, the current theory has to do with the age-old arguments for code (read service) reuse, componentization, interoperability, etc. These arguments totally miss the point of Web services, though. Some claim they want to make these useful services free because it's just a good idea and a cool thing to do for consumers. They usually have the same motives as the open-source camp. And while it's a perfectly valid stance, this claim doesn't ring true on the gut level.

Many people have told me they're interested in exposing functionality from legacy systems out to the World Wide Web and modern types of client programs. That's great, but the question still remains: "Why?"

Why would a company want to spend all that time and effort in creating a Web service for the world to use, free?

As former President Bill Clinton said in 1992, "It's the economy, stupid."

Any company devoting the energy to Web services had better think of how the bottom line gets a boost from this deal. The great thing about exposing an electronic asset (whether legacy or newly developed) to the public as a Web service is that it's a low-cost channel to a new revenue stream – a revenue stream with no hard limits on growth. Making a buck off a valuable service should be the key consideration of any corporate provider. So, what does it take to provide a successful, commercial Web service offering?

Show Me the Money

Cutting to the chase, if you want to make money from a Web service, you need to, first and foremost, figure out who's using your service. You might, in fact, want consumers to request permission before using your service or developers to notify you before embedding your offering within another Web service. Once you know who's using your service, you know where the revenue has to come from.

"Why would a company want to spend all that time and effort in creating a Web service for the world to use, free?"

The next question you need to ask is how much value will users derive from your service. You can figure this any way you like, but at the very least you'll want to know when and how often particular users invoke your service, and whether requests are successfully handled. It's crucial for generating bills and billing strategies to keep track of your customers' usage statistics and patterns. A side benefit of collecting usage information is the enormous value these statistics bring to the process of planning for the future growth of your service.

Almost There But Not Quite

User identification and usage accounting get you 80% of the way to a profitable Web service. But there are a few other considerations that come into play that are fairly important in pulling together a financially successful service. The first thing is quality of service.

You could have a tremendously useful service deployed with a happy, loyal group of customers, but if these customers start to notice consistently slow service or availability problems they're going to drop you like a bad transmission. This is also true if your service returns bogus responses more than once or twice. To present a high-quality Web service, you need to constantly monitor the health of your service in terms of its ability to work through the incoming requests in a timely fashion, the validity of your service's responses, and the availability of your service to the outside world.

And, of course, some living, breathing person in your organization must be notified if things aren't performing up to expectations.

Speaking of the customer experience, let's shift the focus to the developers who will be embedding your service. What happens to them when you're ready to upgrade your Web service? If you change function signatures or add new features, these developers' applications will suddenly be out of date. If you have to take the machine on which your service is hosted offline for maintenance, what happens to all the third-party applications that use your service?

Keep in mind that these events are natural occurrences in the life cycle of any Web service. You need to have some kind of life cycle management support in place to deal with issues of migration, downtime, and upgrades gracefully. Useful error messages, forwarders, notifications of change, and provisioning of documentation about migration paths can keep developers and other users in the loop on the evolution of your service. Lack of care with these eventualities can create a perceived problem with quality of service – causing users to look elsewhere for fulfillment of needed services.

User recognition and tracking, quality of service, and life cycle management are a few areas you can concentrate on to greatly improve your chances of financial success for your Web service. The more you know about your users, the more reliably you fulfill their needs – and the less you irritate them, the better off you'll be.

What are examples of Web services that have these qualities?

'We Have a Problem…'

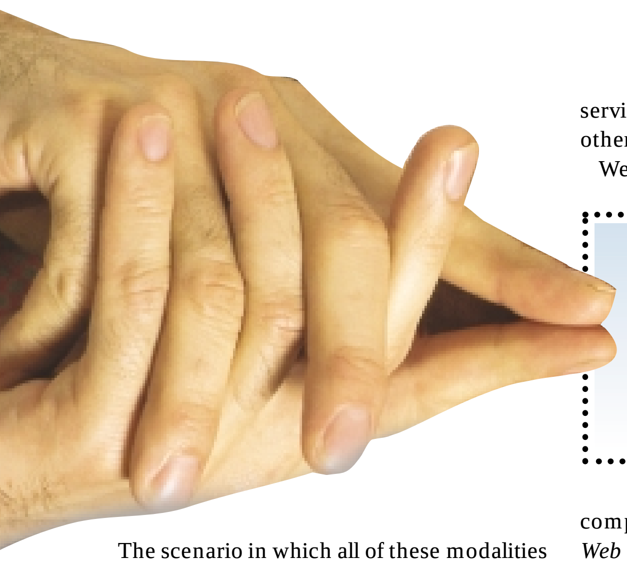
However good all that sounds, the unfortunate facts of life are that few Web services platforms actually support any of these features, and no platform supports them all. This means that when you develop your service you either implement many of these extras or ignore them.

To achieve the kind of Web services addressed, you can't do the latter. Without user recognition and usage tracking, you have no idea whom to bill or for how much. Without some kind of service monitoring and emergency alerts, your users may be denied service unexpectedly. And without a solid system for managing changes, developers who rely on your Web service can have the rug yanked out from under them.

But is it really feasible to implement the necessary infrastructure for all of this any time you want to deploy a Web service? It's true that enterprise providers may already have user databases and change management systems in place, but what kind of effort is involved with bridging these systems into the Web services space? How much work does it take to integrate an enterprise-scale, back-end security system with the average Web services platform? In short, what does it cost to provide the necessary modal features to your Web service to ensure its success?

Software AG

www.softwareagusa.com



The scenario in which all of these modalities must be fully implemented by the provider may turn out to be more costly than the perceived ROI, especially in the short term. Potential providers will take a dim view of exposing Web services if it means devoting too much development effort to ensuring a quality product on top of a financial return.

So if there is functionality lacking in Web services platforms that's also too costly or too inconvenient to develop in-house, how will you get that Web service published as a quality offering that can make you money? Luckily, a solution is staring you in the face …

New Levels of Flexibility

Architecturally speaking, the Web services style of distributed computing provides new levels of flexibility. Partitioning of subsystems and assignment of responsibilities take on a whole new meaning when you consider that specific modal features, or even entire subsystems, could be hosted at another site or outsourced altogether. Web services solutions have the option of taking advantage of other Web services to help get the job done. In other words, your Web service can easily invoke any number of other Web services in the course of fulfilling a request made of it. And this relates directly to your production of a successful Web service.

Doing Only What It's Good At

All of the things addressed so far as important aspects of Web services deployment and still lacking from existing platforms can be (and in at least one case are being) implemented in terms of other Web services. Imagine a small set of Web services, each designed for a specific, narrow purpose. In this type of world, your service should be designed the same way. Each service does only what it's good at and what it's designed for – and outsources the rest of the work. Your service provides just the value it extracts from your legacy system or derives from your business processes, and leaves the rest of the modal concerns to other, better-suited services available in other departments or other companies.

Web services are more than just simple components with SOAP interfaces. The term *Web service* applies to the sum of all of a service's functions, modalities, levels of service, and terms of use. In a scenario such as this, I wouldn't refer to your original component as the Web service but rather as the published, publicly available facade to your service. The full cast of helper services constitutes the actual Web service.

"Web services are more than just simple components with SOAP interfaces."

A Digital-Signature Scenario

You want to offer a service that provides digital signatures for submitted content and stores hashes for inspection much later in case there's some doubt about the validity of the signature. To get this service up and running, you have plenty of development tasks already. You opt to outsource the user tracking and life cycle management services. Using this scheme, here is a high-level view of the typical end-to-end request:

- A new user makes a one-time visit to the preferred authorization service to request credentials and gain access to your service.
- The user invokes the authorize() method of the Web service facade, presenting credentials.
- The Web service in turn invokes the authorization service, validating the user, returning a time-boxed authorization key.
- The key is presented by the user when a function is invoked and used to both identify and track the user.
- When activity is tracked, notifications are sent to the usage auditing service.
- At some point in the future, a billing system will query the usage auditing service for activity records to generate a bill for the users of the service.

"Architecturally speaking, the Web services style of distributed computing provides new levels of flexibility."

Or consider this upgrade situation:

- A week after you put the service up for the first time you realize one of the function signatures needs to be tweaked.
- You contact the life cycle management service and submit an upgrade.
- You stipulate how long the old service should remain active.
- All users and intermediates repackaging your service are notified and informed of the change by the life cycle service.
- A new service facade is created and deployed.
- At the appointed time, the old service is brought down and a forwarder is left in its place.

A Look into the Future

In future articles, I will profile several types of helper services and discuss how each contributes to the web of services collaborating on behalf of your service. I also will describe a framework for non-intrusive integration of the helper services with your services that minimize the amount of development effort involved. Finally, I will elaborate on ways that development and deployment tools could hook into such a Web for even more efficiency at the project scale.

Extending Platform Reach

We've seen how third-party Web services can be leveraged to extend the Web services platforms' reach just as third-party libraries extend the facilities of operating systems. In particular, deployment of stable, profitable Web services could use a little help from some of the helper services I proposed.

In the time since this article was originally written a number of helper services of exactly the kind I described have popped up. Examples include credit card validators, usage trackers, and up-time monitors. Check out your favorite Web service index to watch the groundwork of the service Web being laid!.

Acknowledgment

Special thanks to Courtney Wallace and Russell Alexander.

Attunity

www.attunity.com

Written by Norbert Mikula

Web Services – What Have You Done for Me Lately?

Let's talk dollars and sense

Most of the major providers of development platforms have already started shipping development tools for Web services. It seems inevitable that everyone will jump on the Web services bandwagon in the not-so-distant future.

However, in the minds of many there are still lingering questions: Why bother? What can I do with Web services that I couldn't do before? And maybe even more relevant: How can I do it better?

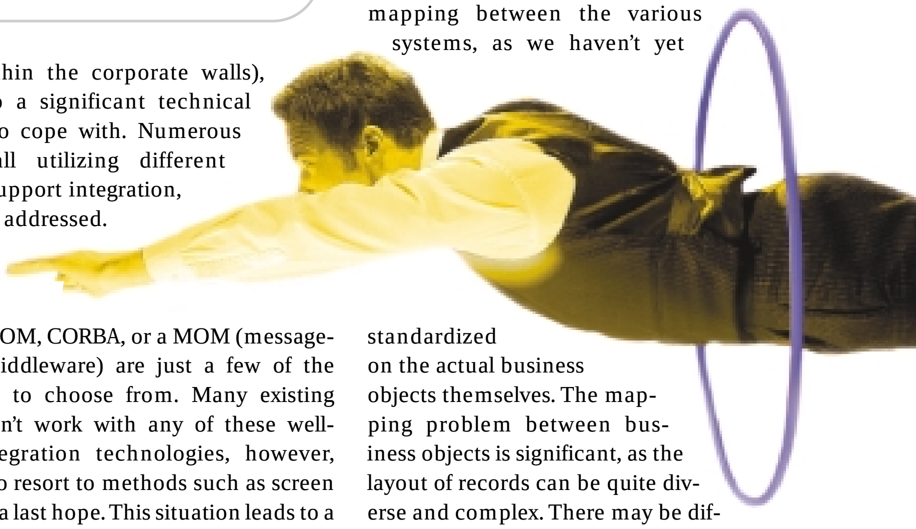
This month we'll examine how Web services can help companies cut costs, create new revenue sources, and ultimately improve their bottom lines. As part of this discussion, we'll reintroduce a term that's once again in the hearts and minds of all IT executives: ROI – Return on Investment. We'll look at both sides of the equation, "return" as well as "investment."

Ultimately, companies will measure ROI for Web services by looking at the various kinds of software packages that utilize them, but this article focuses on the more generic characteristics of the technology itself. Also, instead of looking at distant next-generation business models, we'll focus on the pragmatic realities of today.

The Integration Challenge

Cheaper

Admittedly, integration of heterogeneous and previously unrelated systems is a nightmare, and a costly one at that. Besides the political and cultural issues involved (just staying within the corporate walls), there's also a significant technical challenge to cope with. Numerous systems, all utilizing different means to support integration, have to be addressed.



COM/DCOM, CORBA, or a MOM (message-oriented middleware) are just a few of the approaches to choose from. Many existing systems don't work with any of these well-known integration technologies, however, forcing us to resort to methods such as screen scraping as a last hope. This situation leads to a very diverse back-end integration architecture that's inflexible and expensive to maintain.

The situation for integration across multiple enterprises is even more complex. Work across the extended enterprise needs to overcome the diversity of systems that's a magnitude higher than the already complex case of integration of systems inside the firewall. Integration across corporate boundaries is best accomplished using technology that allows us to use the most pervasive piece of infrastructure we've developed over the course of the last few years – HTTP. Yet this technology should not be dependent on particular transport protocols, offering the flexibility to cooperate with existing infrastructures within enterprises.

Another core technology for next-generation integration is WSDL, the Web Services Definition Language, which allows us to provide a standards-based way of defining our interfaces, including parameter names and valid data types. Instead of dealing with very diverse (and often incompatible) ways of describing interfaces, now there can be one, dramatically reducing the complexity of our integration. Services, described in WSDL format and accessible via Internet-based protocols (such as SOAP/HTTP), will allow us to cut costs considerably from an integration perspective.This might not be seen so much on a small scale, but will certainly be apparent if we envision that our services will be consumed by multiple applications and integrated into the system of multiple business partners (customers or suppliers). Just as with traditional integration broker technologies and their associated ROI metrics, the more interfaces need to be integrated, the more such a universal "access bus" will pay off.

While Web services now let you get to the data, you'll still have to deal with data-mapping between the various systems, as we haven't yet standardized on the actual business objects themselves. The mapping problem between business objects is significant, as the layout of records can be quite diverse and complex. There may be differing and sometimes conflicting business rules in the mix as well. Eventually, this problem will have to be addressed by some generic horizontal standards and more specialized vertical ones. Even then, you still have to map internally to these objects unless you use them already at the core of your schemas – which is unlikely unless you are building the system from scratch.

Faster

Here's where Web services really shine. Mapping problems aside, it takes just a few quick seconds to pick a WSDL file and integrate the respective service into your environment by

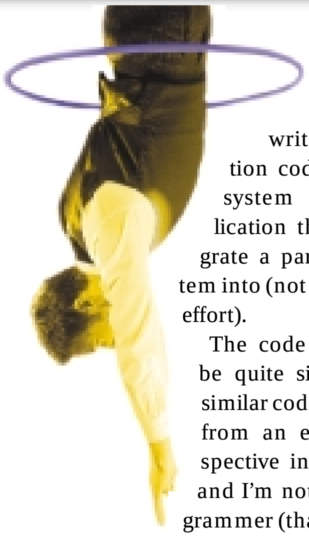
Author Bio

Norbert Mikula is a founding member of DataChannel and an integral part of their strategic product planning and technology research. He has more than 10 years of experience in building and delivering Internet and e-business technologies. Norbert serves as vice-chairman of the board of directors of OASIS and is industry editor of *Web Services Journal*. He is recognized internationally as an expert in Internet and e-business technologies.

NORBERT@DATACHANNEL.COM

AltoWeb

www.altoweb.com



using the appropriate software.

Compare this to writing specific integration code for every back-end system and for every application that you need to integrate a particular back-end system into (not to mention the testing effort).

The code used to do this can be quite simple. I've developed similar code as a piece of software from an enterprise portal perspective in a number of days – and I'm not the world's best programmer (that's why I have to write about ROI).

Return – The Upside Challenge

Customer Satisfaction

As illusive and intangible as the notion of improved customer service may seem, happy customers do translate directly into revenue. Web services will enable us to create new and diversified products and deliver them to our customers in a faster and more cost-efficient way. How? Because, as discussed in the previous section, they allow us to rapidly create new applications that consist, to a large extent, of already existing application services (both internal and external).

Business Integration Across the Extended Enterprise

While the dynamic creation of business relationships through Web services is promising and sounds like an opportunity for exciting new business models, there's another, more immediate benefit to be realized. As a result of their "integrative nature," Web services make it easier for companies to link to their business partner's systems, and in doing so they actually become a part of their partner's core business processes. This holds true regardless of whether the link is a machine-to-machine type of integration or a more interactive portal.

For example, a well-developed set of Web services can be integrated into our customers' intranet and thus further help to establish a long-term and beneficial business relationship further increasing the chance of us, as an "integrated supplier," closing business before others can.

Software as a Service

Lastly, offering software as a service (and charging for it) is going to be an exciting model to look at for revenue generation. This kind of revenue generation requires a separate discussion, though.

Beyond ROI

Especially in a belt-tightening economy, investments will be scrutinized for their exp–ected tangible returns. However, as we look at these calculations we must not forget that infrastructure investments such as Web services also have a strong long-term strategic component that must be taken into consideration.

Increased flexibility in our enterprise backbone, including a library of reusable software components as Web services, will often be hard to measure but should nevertheless also be part of a balanced project proposal and part of a company's long-term competitive plan. The same holds true for the ability to extend the reach of services beyond the browser to cell phones as well as PDAs.

Agility

We live in a dynamic world. Change and new customer requirements are the norm, not the exception. Competitive pressures force us to change our system's designs on a regular basis.

Web services – if done correctly – will provide us with a toolbox of services (internal ones as well as external ones) from which we'll be able to pick and choose the components needed to satisfy new requirements.

Thinking about intranet or extranet portals (just to emphasize again that Web services are not just machine-to-machine technology), the flexibility provided by this new technology will allow us to react to changing requirements and build new or modified portal workspaces in a much more dynamic fashion.

One note of caution, though: this assumes all the services you need are available and are sufficiently parameterized to be used in multiple contexts as well.

Investment

There is no return without investment. The investment required for Web services is potentially considerable. What are some of the factors (besides the associated costs for the software you use)?

Paradigm Shift

In the past we have architected and developed systems in a fairly controlled way. We analyzed a business problem, wrote the specification, and developed to that specification (and hopefully tested along the way). While many of the core objects of the system were often developed with reusability in mind, the degrees of adaptability were limited to the minimal level demanded by the targeted solution.

For Web services to truly work we must develop business objects that are highly parameterized and configurable to support the reuse envisioned by the brave Web services pioneers. Furthermore, we need to keep these services at the level of business objects and services rather than lower-level system services.

In general, Web services may also bring with them a stronger focus on modeling than before. This will result in a need to utilize a new skill set that may be brought into your organization through training or new personnel. So, what's the net? Architects and developers will have to learn new tools and think about new development patterns. This in itself can require major training efforts that are often costly (the training itself plus lost productivity during the time of the training).

Convert Applications to Services

This is where the rubber meets the road. Certainly, the new tools in the marketplace will make it easier to convert existing business objects and functions to Web services. That said, there'll be a few important factors to consider.

Internally, you may have developed some applications you want to convert to the new paradigm. As we've learned through the Y2K experience, reengineering is not as simple as it sounds. At the very least, it's time- and cost-intensive.

Externally, many packaged application vendors will take some time to convert their interfaces to Web services. That also will require upgrading to a new release, which in itself is a costly and time-intensive undertaking (just think about how long it took for many vendors to adopt normal Web-based interfaces).

Conclusion

As with many of the other major paradigm shifts, there will be costs and risks involved to be able to harvest all the returns. The benefits, however, of Web services, tangible or more strategic in nature, warrant an investment in this new technology.

What approach did you use to sell the idea of Web services inside your company? Please e-mail me to let me know.

Sonic

www.sonicsoftware.com

Written by Karsten Januszewski

UDDI and WSDL: Natural Companions

Enabling a software landscape based on Web services

The Web services model relies on a stack of software specifications that makes the interoperability of heterogeneous environments a viable possibility. Two essential specifications of this stack are WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery, and Integration). Understanding each of these initiatives alone is fundamental to comprehending the Web services model; understanding the relationship between the two is critical to developing robust software based on Web services.

Before drilling into the relationship between these two initiatives, it makes sense to address them individually. Each offers powerful software potential in its own right, but their usage together helps create a truly exciting new software ecosystem.

Recipes and Cookbooks

WSDL provides an XML grammar to define the abstract interface and protocol bindings necessary to invoke a Web service. Fundamentally, WSDL describes the contract of that Web service so a client can interact with it. By interpreting a WSDL file, a piece of client software can be written to send and receive messages correctly to that Web service. You might say that WSDL represents the "recipe" of a Web service. From an object-oriented language perspective, it's analogous to an IDL file.

UDDI is a group of Web-based registries designed to house information about businesses and services in a structured way. Through UDDI, you can publish and discover information about a business and the services it provides. This data can be classified using standard taxonomies so that information can be discovered based on categorization. Of primary importance is that UDDI contains information about the technical interfaces of a business's services. Through a set of SOAP-based XML API calls, you can interact with UDDI at both design-time and runtime to discover technical data about a business's services, so those services can be invoked and used.

If WSDL files are a kind of recipe for interacting with Web services, then UDDI might be referred to as the cookbook where all the recipes reside. It offers a place to publish and inquire about different services based on different classifications and protocols.

However, this analogy is potentially inaccurate and misleading. A common misconception about UDDI is that it's nothing more than a cookbook – a massive White and Yellow Pages for Web services. While these analogies have some validity, they undercut and reduce UDDI's scope and intention. UDDI was designed to accomplish goals beyond simply offering a Web directory of services.

UDDI is a registry in the sense that it contains entries that direct users to resources outside of UDDI. The data it contains is relatively lightweight; as a registry, its task is to provide network addresses to the resources – schemas, interface definitions, endpoints – in other locations.

UDDI has been designed in a highly normalized fashion, not bound to any technology. In other words, a UDDI registry entry can contain any type of resource, whether that resource is XML-based or not. For example, the UDDI registry could contain information about a business's EDI system, its DCOM interface, or a service that uses the fax machine as its primary communication mechanism. The point is, while UDDI itself uses XML to represent the data it stores, it allows for other kinds of technology to be registered. This design decision makes UDDI an enduring piece of infrastructure that won't become outdated as technology moves forward.

Beyond the Cookbook: Interfaces and Implementations

A consideration of interfaces and implementations helps us grasp how UDDI and WSDL work together. Both WSDL and UDDI were designed to delineate clearly between abstract metadata and concrete implementations. Understanding the implications of the division between abstraction and implementation is essential to understanding the design decisions behind WSDL and UDDI.

For example, WSDL makes a clear distinction between *messages* and *ports*. Messages, the required syntax and semantics of a Web service, are always abstract, while ports, the network address where the Web service can be invoked, are always concrete. You're not required to provide port information in a WSDL file. A WSDL file can contain solely abstract interface information and not provide any concrete implementation data. Such a WSDL file is considered valid. In this way, WSDL files are decoupled from implementations.

Perhaps one of the most exciting implications of this is the possibility of multiple implementations of a WSDL interface. This design engenders the opportunity for disparate systems to write implementations of the same interface, guaranteeing that the systems can talk to one another. If three different companies have implemented the same WSDL file and a piece of client software has created the proxy/stub code for that WSDL interface, then the client software can

Author Bio
Karsten Januszewski is a program manager for UDDI within Microsoft's Business Applications Division, and a member of the UDDI Working Group. His background includes field service management, application service provider software, and B2B supply chain management software. Karsten has also worked for the Microsoft Healthcare Users Group, meeting the needs of information systems developers and users in the healthcare industry.
KARSTENJ@MICROSOFT.COM

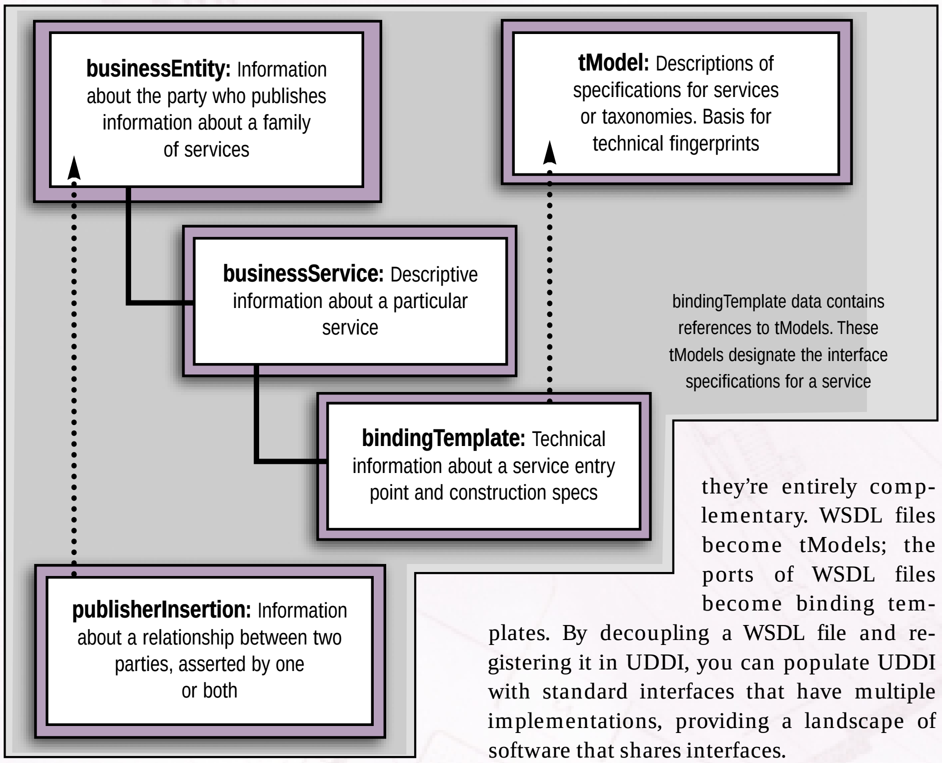

businessEntity: Information about the party who publishes information about a family of services
tModel: Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
businessService: Descriptive information about a particular service
bindingTemplate data contains references to tModels. These tModels designate the interface specifications for a service
bindingTemplate: Technical information about a service entry point and construction specs
publisherInsertion: Information about a relationship between two parties, asserted by one or both

FIGURE 1 | **Binding template and tModel**

communicate with all three of those implementations with the same codebase.

UDDI draws a similar distinction between abstraction and implementation with its concept of tModels. The tModel structure, short for "Technology Model," represents technical fingerprints, interfaces, and abstract types of metadata. A corollary of tModels is the binding template, which is the concrete implementation of one or more tModels. Inside a binding template, you can register the access point for a particular implementation of a tModel. Figure 1 demonstrates this relationship.

Just as the schema for WSDL allows you to decouple interface and implementation, UDDI provides a similar mechanism. tModels can be published separately from binding templates that reference them. For example, a standard's body or industry group might publish the canonical interface for a particular use case or vertical, and then multiple businesses could write implementations to this interface. Accordingly, each of those business's implementations would refer to that same tModel.

Because both the UDDI and WSDL schema have been architected to delineate clearly between interface and implementation, the two constructs work together naturally. In fact, they're entirely complementary. WSDL files become tModels; the ports of WSDL files become binding templates. By decoupling a WSDL file and registering it in UDDI, you can populate UDDI with standard interfaces that have multiple implementations, providing a landscape of software that shares interfaces.

Example

There are many software scenarios where this environment might manifest itself. One example is in the electric commerce arena. Today, anyone who's ever been involved in an interoperability project knows that most of the work is in getting the two systems to talk, agreeing on protocols, transport mechanisms, and so on. With WSDL, the contract is defined semantically, simplifying interoperability enormously. And by using UDDI to discover multiple implementations of that WSDL, an entire trading community can be tapped into, based on UDDI queries. Moreover, by writing UDDI inquiry into its software, a company could dynamically and programmatically query UDDI at runtime to update information about a trading partner's Web service.

A useful example of how UDDI and WSDL relate can be found by looking at the UDDI Web service itself. The UDDI API is a SOAP-based XML API defined by an XML Schema (www.uddi.org/schema/2001/uddi_v1.xsd) and corresponding WSDL files for inquiry (www.uddi.org/wsdl/inquire_v1.wsdl) and publishing (www.uddi.org/wsdl/publish_v1.wsdl). If you examine these WSDL files, you'll discover they don't have a <service> element identified; in other words, no endpoints are provided.

These two WSDL files have been registered in UDDI as tModels. For instance, to see the uddi-org:publication WSDL defined in UDDI, see Listing 1. It's worth noting here:

1. The tModel has a tModelKey associated with it in the convention of a uuid. UDDI generates a uuid for every entity registered in it; this allows for the ability to uniquely refer to this UDDI registry entry in other entries, like a foreign key.
2. The contents of the <overviewURL> correspond to the location of the WSDL file hosted at www.uddi.org. Again, UDDI is a registry that directs the user to resources outside of UDDI.
3. This tModel has been categorized using keyedReferences in the <categoryBag> element. Each of the keyedReferences corresponds to a canonical taxonomy, as specified in the UDDI API specification (www.uddi.org/specification.html). These categorizations allow a human or a machine to know some details about this tModel – in this case, the most important categorization defines this tModel of type "wsdlSpec". By following these conventions, a user can write tools that can search UDDI and find only tModels that are categorized in a certain way.

"If WSDL files are a kind of recipe for interacting with Web services, then UDDI might be referred to as the cookbook where all the recipes reside"

If you're interested in finding implementations of this tModel, you could issue the following query to UDDI, which would search for any businesses that have implemented that particular tModel:

```
    <find_business xmlns="urn:uddi-
org:api" generic="1.0">
       <tModelBag>
          <tModelKey>uuid:64c756d1-3374-
4e00-ae83-ee12e38fae63</tModelKey>
```

```
    </tModelBag>
  </find_business>
```

> "UDDI and WSDL act as complementary specifications that assist in enabling a software landscape based on Web services."

The uuid of the uddi-org:publication tModel is passed in the tModelBag of the find_ business query. The natural language version of this query states, "Find all businesses that have implemented the uddi-org:publication interface."

Issuing such a query to UDDI today would return IBM and Microsoft, the two companies that currently host UDDI operator nodes. In the future, as other operator nodes emerge, this query would return multiple results. Once the <find_business> result set is obtained, you can drill down further into the service information of Microsoft and IBM and discover the accessPoint for the UDDI Web service. A fragment of the Microsoft XML that contains the UDDI Web service is shown in Listing 2.

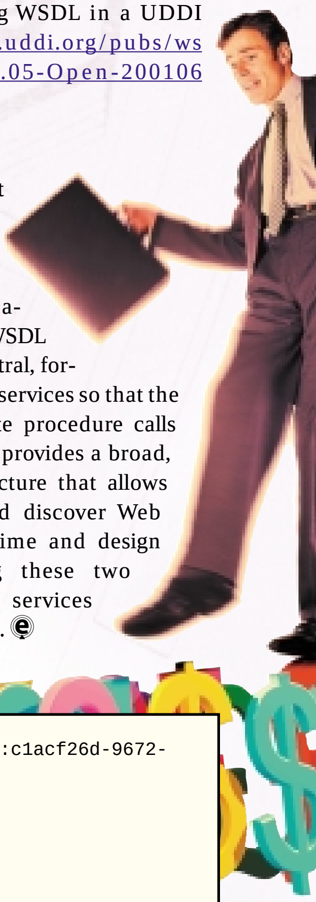
Moving Full Circle

A few things to point out: in the <bin ding Template> is an <accessPoint> where the Web service can actually be invoked. Also, a child of the <binding Template> is the <tModel Inst anceDet ails>, which refers to the uuid that contains the WSDL for the uddi-org:publication tModel. The link between interface and implementation is created in the tModel InstanceDetail. Thus, we have moved full circle from interface definition to implementation specifics.

To correctly register WSDL in UDDI, follow the set of best practices defined in the document "Using WSDL in a UDDI Registry 1.05"(www.uddi.org/pubs/wsdlbestpractices-V1.05-Open-20010625.doc).

Conclusion

UDDI and WSDL act as complementary specifications that assist in enabling a software landscape based on Web services. WSDL provides a vendor-neutral, formal way to define Web services so that the next-generation remote procedure calls can be realized. UDDI provides a broad, standardized infrastructure that allows users to describe and discover Web services, both at runtime and design time. By combining these two standards, a Web services ecosystem can flourish.



Listing 1

```
   <tModelDetail generic="1.0" operator="Microsoft
Corporation" truncated="false" xmlns="urn:uddi-org:api">
      <tModel tModelKey="uuid:64c756d1-3374-4e00-ae83-
ee12e38fae63" operator="www.ibm.com/services/uddi"
authorizedName="0100000M99">
        <name>uddi-org:publication</name>
        <description xml:lang="en">UDDI Publication API -
Core Specification</description>
        <overviewDoc>
          <description xml:lang="en">This tModel defines
the publication API calls for interacting with the UDDI
registry.</description>

<overviewURL>http://www.uddi.org/wsdl/publish_v1.wsdl</over
viewURL>
        </overviewDoc>
        <categoryBag>
          <keyedReference tModelKey="uuid:c1acf26d-9672-
4404-9d70-39b756e62ab4" keyName="types"
keyValue="specification" />
          <keyedReference tModelKey="uuid:c1acf26d-9672-
4404-9d70-39b756e62ab4" keyName="types" keyValue="xmlSpec"
/>
          <keyedReference tModelKey="uuid:c1acf26d-9672-
4404-9d70-39b756e62ab4" keyName="types"
keyValue="soapSpec" />
          <keyedReference tModelKey="uuid:c1acf26d-9672-
4404-9d70-39b756e62ab4" keyName="types"
keyValue="wsdlSpec" />
        </categoryBag>
      </tModel>
    </tModelDetail>
```

Listing 2

```
     <bindingTemplate serviceKey="33c3d124-e967-4ab1-8f51-
d93d95fac91a" bindingKey="48f2bc6b-a6de-4be8-9f2b-
2342aeafaaac">
            <description xml:lang="en">Production UDDI
server, Publishing interface</description>
            <accessPoint
URLType="https">https://uddi.microsoft.com/publish</accessP
oint>
            <tModelInstanceDetails>
              <tModelInstanceInfo
tModelKey="uuid:64c756d1-3374-4e00-ae83-ee12e38fae63">
                <description xml:lang="en">UDDI SOAP
Publication Interface</description>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
     </bindingTemplate>
```

Download the code at sys-con.com/webservices

Written by Andy Longshaw

Exploding Web Services Myths

Unrealistic expectations may be the real villain

I was recently sketching out an overview of Web services for a colleague, including SOAP, UDDI, and WSDL. After a few minutes he came to the conclusion, "Ah, it's like CORBA, but using XML and HTTP." While this may have reflected the paucity of my description, it did seem to echo the opinions I heard "on the ground" talking to developers who are exposed to a variety of messages about Web services.

Some dangerous assumptions are being made about Web services. This is partly due to the origins of some of the technologies involved and partly to the hy-perbole surrounding this new movement in the creation of distributed systems. Even discounting any blue sky gazing about "smart" Web services, here I consider what type of application can actually be built with the Web service technologies available today.

Exploding the RPC Myth

Human beings like to compartmentalize things. We're comfortable if we can put things in a familiar "box." This is why analogy is a great aid to learning. When most people who are familiar with traditional distributed component systems see the combination of SOAP, UDDI, and WSDL, they will readily match them up with RPC, name services, and IDL respectively. This leads them, however, to think of this model of Web services as a dynamic mechanism.

It's easy to assume that applications using this model will look up the price of a commodity (for example, steel) on a daily, or even minute-by-minute, basis and will swap suppliers dynamically based on these figures. This assumption is aided and abetted by many previous visions of dynamic markets put forward as part of a future Internet.

The reality is that markets like these won't happen in the short term. Instead, the SOAP/UDDI/WSDL combination will be used to improve existing ways of doing business. Existing Web/EDI links and prototypical (i.e., do-it-yourself) Web services will be replaced by standard Web service mechanisms. However, relationships will be set up at human speeds by human application developers and decision makers. The central determinants of the business relationship, such as the quality of the product and the trustworthiness of the supplier, will still be made in human terms.

This isn't to say that some markets won't use Web services dynamically — finance being the obvious one. Financial data already occupies a dynamic world with fortunes being made and lost on the availability of such data. An ultra-dynamic model of Web services will work well in this arena since it's simply mapping existing practices. However, many markets won't see a benefit from such dynamic Web services.

Will a company change its supplier of ball bearings to a relatively unknown company across the far side of the world based on a lower price found from a business directory? In most markets, any form of dynamic change or selection will occur within the context of a "local community" of suppliers. This parallels the way people buy certain things through the Internet, but still buy most things primarily from their local shops.

Rejecting the Global Registry

Another dangerous assumption that seems to be made by many people when they first look at Web services is the idea of the global registry. This distributed registry will contain all of the information required to find and interact with businesses throughout the world, thus enabling the ultra-dynamic Web services discussed earlier. This assumption may come about through our familiarity with the global nature of the Internet and particularly its naming service (DNS).

In many ways, DNS is a wonder of the modern world. It is one of the underpinnings of the Internet and it works fairly seamlessly; yet it's the shared responsibility of many organizations with no formal relationships between them. If DNS works, then why not have a global business registry along the same lines? There are a number of answers to this question:

- DNS exists because it passes small amounts of data and caches well. The DNS data passed around consists of machine names and IP addresses – in the order of magnitude of 40 bytes per record. The data required to identify, select, and interact with a business service will be an order of magnitude or two greater than this, even with good compression. This amount of data being passed back and forth could help bring the current Internet to a standstill.
- Given the amount of data in a global registry, can we really search it all using current technology? The searching of data is similar to cryptography in that it benefits from both increasing processor power and improvements in algorithms. Although Moore's law still drives processing power, some algorithmic-level improvements are needed in order to deliver such searching capability. Would it really be able to process all of this data and respond in "real time" for those people who need that level of response?
- DNS is centrally or commonly owned and is based on simple, public standards. Registry services are one area where standards are not yet settled (for example, UDDI versus ebXML Registry and Repository), making the exchange of data far more involved.
- Who would own and run the system? And most important, who would guarantee the searches were "fair"? No one really gains

Author Bio
Andy Longshaw is an independent consultant, writer, and educator specializing in J2EE, XML, Web-based technologies, and components – particularly the design and architecture decisions required to use these technologies successfully. Longshaw has been explaining technology for most of the last decade as a trainer and in conference sessions.

ANDY@BLUESKYLINE.COM

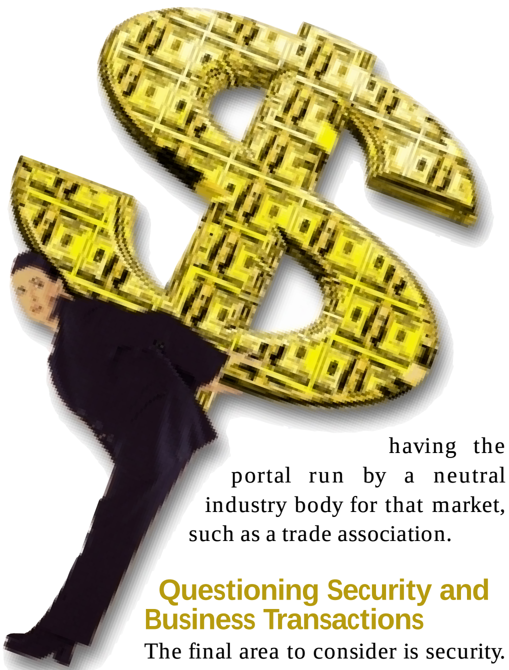
Data Mirror

www.datamirror.com/resourcecenter

by returning his or her own IP address in place of the one searched for in DNS (apart from a few hackers). However, being the first match in a business registry could be of huge benefit in a Web services market. This means that the owners of such registries would be tempted to sell high positions on the list for maximum gain. In that respect, business registries would have more similarities with public Internet search engines.

In my opinion, there will be no global registry – at least not in the short term. What we will see is "local communities" of producers and consumers in vertical markets gathering around "portals." These portals will consist of UDDI registries (or ebXML registries and repositories) containing business data for that specific vertical market.

The provider of the portal will be able to deliver added value by providing some (potentially) trusted rating of the quality and reliability of the suppliers listed. The issue of "rigging" the searches may be solved by a certain degree of transparency, such as noting that particular companies have paid for premium entries or by having the portal run by a neutral industry body for that market, such as a trade association.



Questioning Security and Business Transactions

The final area to consider is security. Security is currently a weak part of the Web services story. Talk of security centers largely around the use of secure sockets (SSL) to pass HTTP requests from client to server and back. This provides a basic level of authentication and privacy between two servers. However, this is not enough for a global e-business infrastructure.

Consider the context for the discussion of global Web services, namely business-to-business transactions, potentially through an intermediary. In the "real" world, orders are authenticated by having signed pieces of paper that can be produced when required to provide a level of nonrepudiation for those transactions. Unless the e-business infrastructure can provide the digital equivalent of this, namely a digitally signed payload for each business message, then a free market in e-business will be seriously hampered. There must be a way to prove the authenticity of the original document, even if it has passed through several intermediaries.

As with most of the other issues stated above, the problem is more one of timing than technical capability. Efforts are afoot to add standard mechanisms of digital signing for XML documents. However, everyday, global use of this will rely on consistent and ubiquitous public key infrastructure that is itself currently proving somewhat elusive.

Going beyond the traditional EDI-style e-business models, people talk of replacing applications with globally accessible Web services that charge per-use rather than a on-off charge. The need for ubiquitous authentication and nonrepudiation is vital for this model since – if you can't identify the user in a nonrepudiable way – how can you charge them for use of the service?

A final aspect in the area of business transactions is the legal basis of the transaction. If something does go wrong, whose jurisdiction does it fall under? In the world of paper contracts, this is usually written into them. For business to proceed, the same type of information must be built into electronic agreements. Even ebXML, with its Collaboration Protocol Profiles and Agreements, doesn't provide this sort of information as part of the negotiated business agreement, although it goes far beyond the simple RPC model discussed earlier in this article and takes us to a different level of binding and services.

Another issue is that ebXML doesn't form part of the most widely promoted model of Web services, namely that based around SOAP/UDDI/WSDL.

The Rush to Web Services

The philosopher George Santayana said, "Those who cannot remember the past are condemned to repeat it." In the early 1990s, the Open Software Foundation developed the Distributed Computing Environment (DCE). DCE was a rich and powerful distributed application environment providing the security and registry services required for truly distributed applications. However, DCE was perceived as too much of a heavyweight for most applications and hardware at the time, so its main legacy is the RPC mechanism that underpins Microsoft's Distributed COM. Having built basic connectivity on these foundations, Microsoft has spent much of the past five years adding in the security and registry services that were left behind in the original DCE.

The same type of issues seem to be facing ebXML – a powerful, XML-based e-business framework that addresses issues such as security and business process integrations. This is seen as being a heavyweight in Web services terms and is in danger of being sidelined (much like DCE was). However, if we don't learn the lessons of the past, we may well have to spend the next few years reinventing the parts of ebXML, and similar initiatives, that are discarded in the rush to Web services. The other side of the coin is that we may be able to survive with minimal services built on the simple building blocks. After all, the Web has come a long way on HTML and HTTP.

The real danger in all of this is that Web services are undermined by the weight of unrealistic expectations. The popular Web service technologies provide a way for us to get from A to B in a flexible and platform-independent way. The benefits of this shouldn't be underestimated, but we're a long way yet from a global e-commerce infrastructure geared around "smart" Web services.

Resources

A list of different ways that UDDI registries may be applied:

1. www-106.ibm.com/developerworks/webservices/library/ws-rpu1.html
2. JamCracker positioning paper on Web services, including registry ecosystems: www.itml.org/whitepapers/jamcrackerwebservicespositionpaper_wp_cust_se_0316011.pdf

Shinka

www.shinkatech.com/register7/

written by Robert McGarvey

STREET FIGHTING Comes to WEB SERVICES

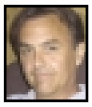
Get ready, because soon the big knock will be at your door and your boss will be standing there with a single question for you: Should we go with .NET or J2EE for our Web services?

Know a couple of things right off the bat. Big bucks will ride on your answer because, whichever direction your company takes, investment in a Web services platform will represent a substantial IT budget commitment. The other fact is that – smile as you might at what seems the improbable pairing of your boss with a cutting-edge technology question – both Microsoft (with .NET) and the Java community (Sun, Oracle, BEA, and others that have focused on J2EE) are flexing their muscles in a massive publicity campaign that is geared towards prodding high-level corporate executives into making commitments on one side or the other. Get ready because he or she will be pounding at your door soon, mainly because their boss probably will be prodding them for insights into the .NET vs. J2EE streetfight.

Higher-up executives will be keenly watching this decision-making process because they have picked up the buzz that the choice made here may fundamentally shape the enterprise's direction in the coming years. Listen to what Eric Rudder, a Microsoft senior vice president for developer and platform evangelism, said in an interview at Microsoft's Professional Developers Conference in October: "For companies, .NET and the world of XML Web services offer the opportunity to transform the way they do business."

or J2EE ?

Author Bio:



Robert McGarvey has covered the Web since 1994 for magazines ranging from "Technology Review" to "Upside." He is the author of the best-selling book "How To DotCom" and a contributing writer to various SYS-CON publications, including Wireless Business & Technology and Web Services Journal.

RJM@MCGARVEY.NET

SHAPING THE FUTURE OF THE ENTERPRISE

That's a mouthful because Rudder is saying this isn't simply an IT bet; it's a wager on how the enterprise should do business. And know that Java evangelists say much the same about J2EE – which means this decision is one that may put business's future on the line.

The fight intensifies because while both .NET and J2EE will get Web services managers where they need to go, there are huge differences between the solutions that, probably, will make one or the other ideal for a given organization. This dust-up isn't about cosmetics; it's about fundamental technological and business practices because J2EE and .NET are wildly different platforms. Where are the differences?

Making the Choice

Hold on for the answers but recognize at the starting point that the cautious choice for most Web services managers right now is to go with J2EE – if, that is, you want to run with the crowd. Analysts project a big lead for J2EE in enterprise installations throughout 2002, in part because J2EE comes into this race with a multi-year lead in deployments. .NET may be from Microsoft but it's the new kid on the block and, entering Q4 2001, it was still in beta, with no firm dates set for full release. Put those factors together and it's not surprising that Randy Heffner, a vice president at Giga Information Group, says that J2EE probably has a "3 to 1 lead" over .NET inside of the enterprise. Looking ahead, Heffner definitely sees .NET closing that gap ("Microsoft continues to make notable improvements in .NET," says Heffner) but at this moment in time the emphatic leader in large-scale organizations is J2EE by a landslide.

Adds John Magee, senior director of Oracle 9*i*: "J2EE is running 3 to 1 ahead of .NET. inside enterprise."

David Bernstein, CTO of enterprise apps developer Chordiant Software, says much the same: "Not one customer has asked us about .NET. We are neutral on this. If a customer asked for .NET, we'd have no problem with the request. But nobody is asking."

Should these votes nudge you into the J2EE camp? Not so fast. Microsoft can never be counted out of any fight so quickly and, for sure, the noise from top executives in Redmond, WA, is that Microsoft is staking a big chunk of the company's future on its .NET initiative. In an interview at that October PDC, Bob Muglia, group vice president of the .NET services group, said, ".NET is Microsoft's platform for XML Web services. It is the foundation for the next generation of software." The unmistakable message is that Microsoft is determined to bring .NET up to parity with J2EE in terms of features, performance, and installed base. Then, too, when the Redmond behemoth stakes out a position, smart Web services professionals sit up and take notice. "Microsoft is Microsoft. They will be there in this fight," says Pete Conner, a CTO at IT consultancy Primitive Logic and J2EE systems integrator.

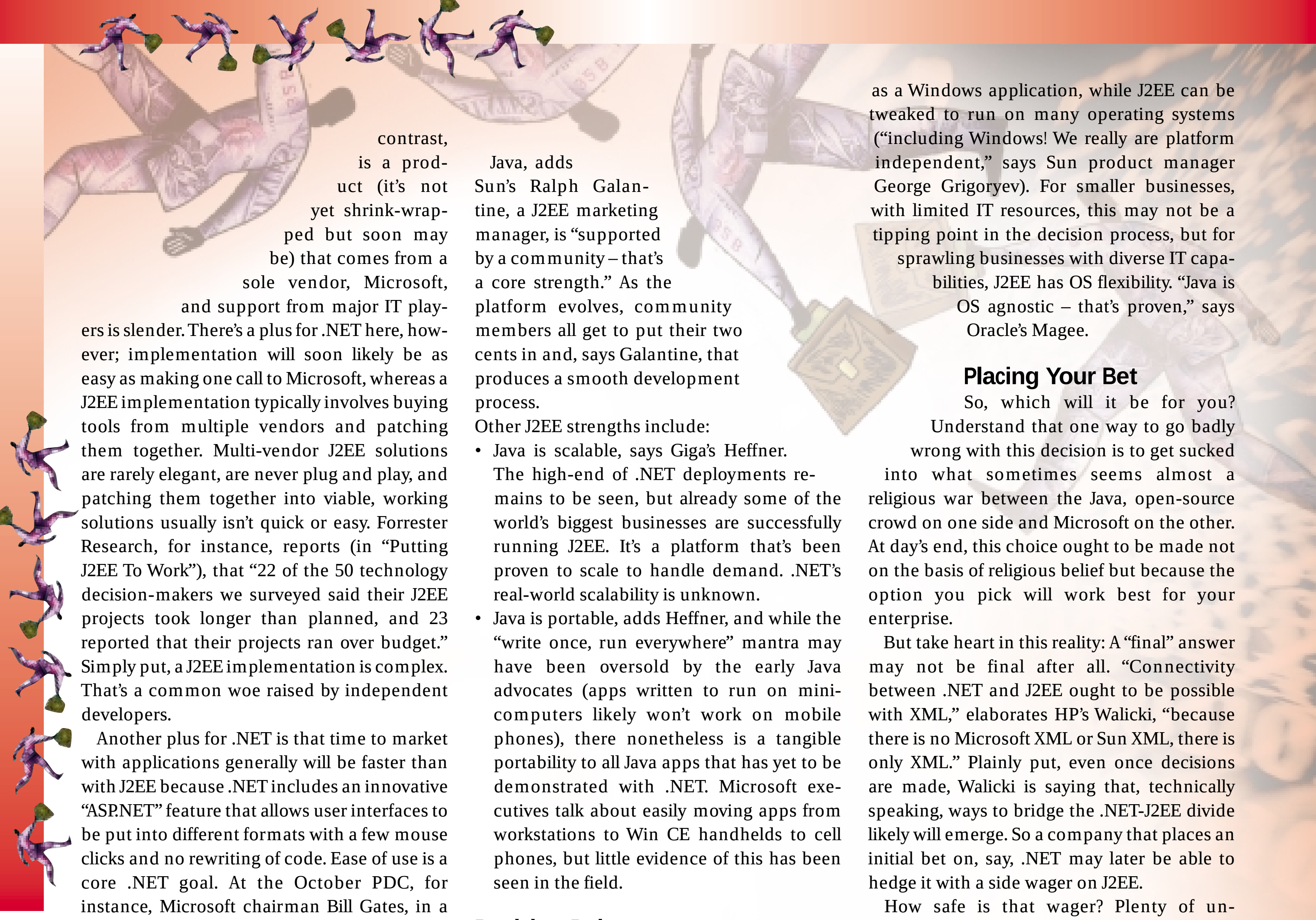
Chew on this too: "The reality is that both .NET and J2EE will get users," says Jack Walicki, general manager of HP Web Services Operations. "It's no leap of faith to envision a company building services on both platforms." Even so, Walicki acknowledges that most businesses probably will standardize on either .NET or J2EE (HP, he says, uses J2EE "because that's the tool we were given"). To make this choice shrewdly, Walicki urges, "The key is to start by understanding what you've done so far, what technologies are you now using, and what do you want to accomplish with a Web services platform?"

"With our customers," he adds, "we don't start by telling them what to do or which to pick. We begin by asking them questions, to figure out where they are and where they want to get to. That's how the best choices will emerge."

A big issue, say the consultants, is that an enterprise that is already running plenty of UNIX or LINUX boxes and has attained a level of comfort with Java, probably will tilt towards J2EE. It's a proven tool for extending legacy systems and applications into Web services. Conversely, a business that is Microsoft-centric – particularly one that's looking to extend the desktop into Web services – probably will want to give .NET a hard look. Are these rules of thumb decisive? Hardly; they're starting points for arriving at a decision, say the analysts, and a lot more investigation needs to be done before any significant enterprise reaches a standardization decision.

Sound off for .NET

Just how do J2EE and .NET differ? J2EE, for instance, is a standard supported by upwards of 18 heavyweight IT players (BEA, IBM, Borland, Sun, Oracle, HP, and many others sell tool sets for implementing J2EE). .NET, by



contrast, is a product (it's not yet shrink-wrapped but soon may be) that comes from a sole vendor, Microsoft, and support from major IT players is slender. There's a plus for .NET here, however; implementation will soon likely be as easy as making one call to Microsoft, whereas a J2EE implementation typically involves buying tools from multiple vendors and patching them together. Multi-vendor J2EE solutions are rarely elegant, are never plug and play, and patching them together into viable, working solutions usually isn't quick or easy. Forrester Research, for instance, reports (in "Putting J2EE To Work"), that "22 of the 50 technology decision-makers we surveyed said their J2EE projects took longer than planned, and 23 reported that their projects ran over budget." Simply put, a J2EE implementation is complex. That's a common woe raised by independent developers.

Another plus for .NET is that time to market with applications generally will be faster than with J2EE because .NET includes an innovative "ASP.NET" feature that allows user interfaces to be put into different formats with a few mouse clicks and no rewriting of code. Ease of use is a core .NET goal. At the October PDC, for instance, Microsoft chairman Bill Gates, in a dramatic demonstration of .NET capabilities, used ASP.NET to build a Web service, then tested it, created a Windows front-end for it, and deployed it, all inside of 15 minutes.

A third plus for .NET: There are significant performance advantages over J2EE (which remains slow in its execution). Page load times are blisteringly fast say .NET advocates, and even J2EE fans admit that a speed-up in Java's performance is sorely desired. "A J2EE sore spot is performance; it's not where it needs to be," says Steve Baker, program manager for GraphON Corporation, a developer of Web infrastructure software.

Jumping Java Beans

Good as the pro-.NET arguments sound, the arguments in favor of J2EE are equally numerous. A key, says Frank Slootman, VP of product solutions at Borland, is that J2EE has been around for a half-dozen years, it's stable, proven technology that is supported by multiple vendors and, therefore, there is no vendor lock-in. For users, this means they can shop around, solicit bids from various vendors, kick a lot of tires, and keep hunting until exactly the right vendor and solution emerge.

Java, adds Sun's Ralph Galantine, a J2EE marketing manager, is "supported by a community – that's a core strength." As the platform evolves, community members all get to put their two cents in and, says Galantine, that produces a smooth development process.

Other J2EE strengths include:

- Java is scalable, says Giga's Heffner. The high-end of .NET deployments remains to be seen, but already some of the world's biggest businesses are successfully running J2EE. It's a platform that's been proven to scale to handle demand. .NET's real-world scalability is unknown.
- Java is portable, adds Heffner, and while the "write once, run everywhere" mantra may have been oversold by the early Java advocates (apps written to run on minicomputers likely won't work on mobile phones), there nonetheless is a tangible portability to all Java apps that has yet to be demonstrated with .NET. Microsoft executives talk about easily moving apps from workstations to Win CE handhelds to cell phones, but little evidence of this has been seen in the field.

Decision Points

Persuaded that there genuinely are big differences between J2EE and .NET? Hold on because those differences are about to get bigger still. Case in point: "Security may be the deciding factor when it comes to choosing which to implement," says Baker. He adds, "Security issues need to be resolved with .NET."

J2EE is designed, from the ground up, to afford extremely high levels of security and the industry is in wide agreement that J2EE succeeds. Microsoft, meantime, has offered assurances that .NET will deliver iron-clad security but even so, says Baker, "Security has to be a question mark with Microsoft." Hackers and virus writers, of course, have had field days with various Microsoft products and while that's not a conclusive indictment (they've probably put more effort into cracking Microsoft products), it's cause for pause before jumping into a full-fledged .NET implementation before the full security story is known and thoroughly tested.

But a still larger question mark looms: Will .NET offer cross-platform compatibility? Microsoft, again, assures that compatibilities will be wide, but so far .NET stacks up entirely as a Windows application, while J2EE can be tweaked to run on many operating systems ("including Windows! We really are platform independent," says Sun product manager George Grigoryev). For smaller businesses, with limited IT resources, this may not be a tipping point in the decision process, but for sprawling businesses with diverse IT capabilities, J2EE has OS flexibility. "Java is OS agnostic – that's proven," says Oracle's Magee.

Placing Your Bet

So, which will it be for you? Understand that one way to go badly wrong with this decision is to get sucked into what sometimes seems almost a religious war between the Java, open-source crowd on one side and Microsoft on the other. At day's end, this choice ought to be made not on the basis of religious belief but because the option you pick will work best for your enterprise.

But take heart in this reality: A "final" answer may not be final after all. "Connectivity between .NET and J2EE ought to be possible with XML," elaborates HP's Walicki, "because there is no Microsoft XML or Sun XML, there is only XML." Plainly put, even once decisions are made, Walicki is saying that, technically speaking, ways to bridge the .NET-J2EE divide likely will emerge. So a company that places an initial bet on, say, .NET may later be able to hedge it with a side wager on J2EE.

How safe is that wager? Plenty of uncertainties need to be resolved before the interoperability of .NET with J2EE can be pronounced as a fact – but encouraging news for any Web services developer is that Web services themselves may prove to be the way out of the .NET-J2EE divide. At least that's the opinion of Annrai O'Toole, a cofounder of Irish middleware company IONA who now serves as executive chairman of Cape Clear, a Web services developer. He explains, "Web services offers the industry a solution to the bickering that has created 20 years of IT incompatibility." How? O'Toole goes on, "The arrival of widely accepted standards such as XML provides a common base platform that supersedes religious arguments over operating systems, languages, tools, and applications. Web services prepares the ground for a new era of cooperation, if you will: a third way."

"Web services," adds Conner, "has a huge future" – and a big slice of that future probably will be in creating ways for J2EE and .NET to interact and exchange information. Go ahead, tell your boss that. In other words, now is definitely the time to begin a sizable Web services commitment in order to deal with the fallout of the J2EE-.NET dust-up.

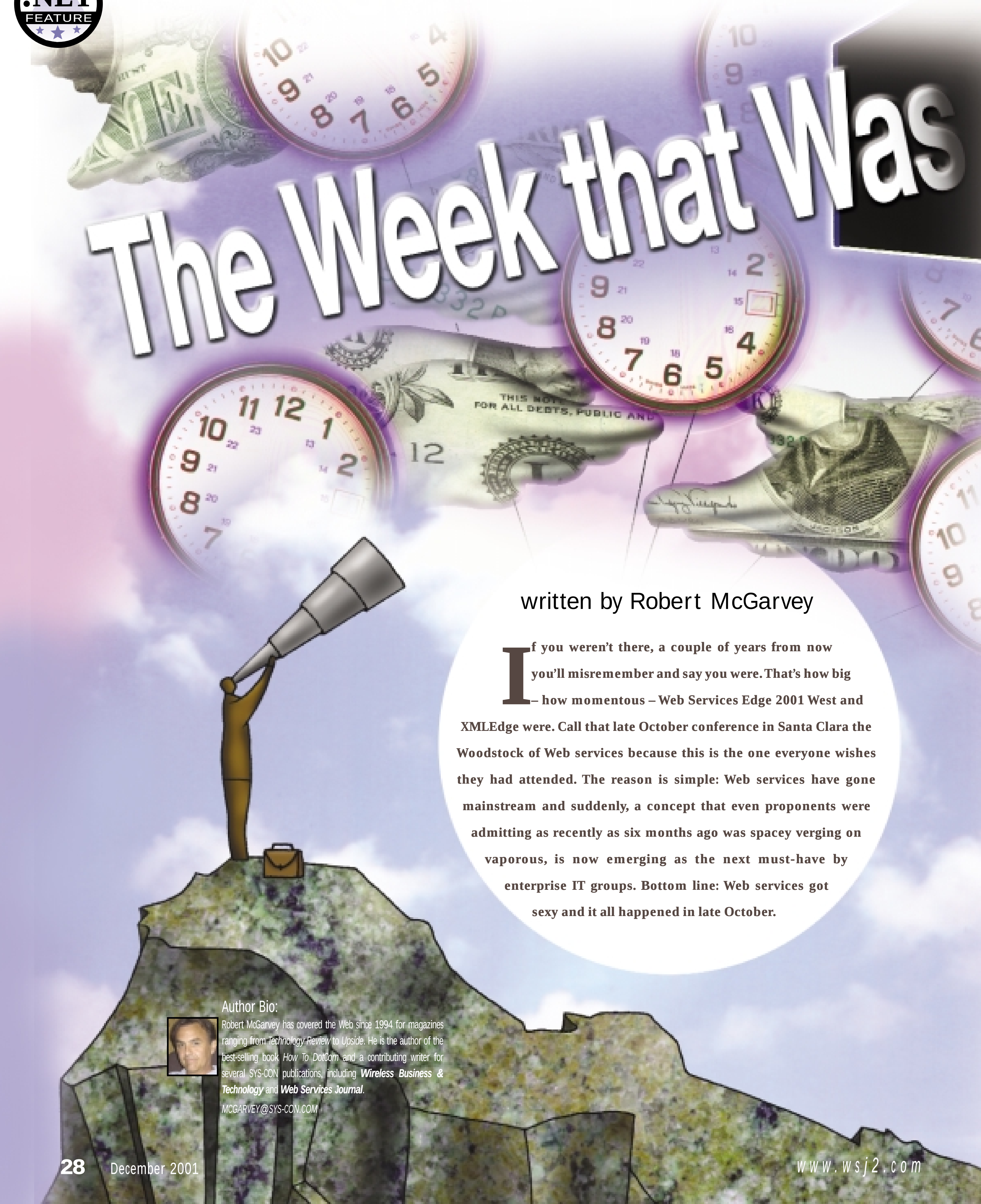
Infragistic

www.infragistic.com

.NET FEATURE

THE WEEK OF OCTOBER 22 IN WEB SERVICES HISTORY!

The Week that Was



written by Robert McGarvey

If you weren't there, a couple of years from now you'll misremember and say you were. That's how big – how momentous – Web Services Edge 2001 West and XMLEdge were. Call that late October conference in Santa Clara the Woodstock of Web services because this is the one everyone wishes they had attended. The reason is simple: Web services have gone mainstream and suddenly, a concept that even proponents were admitting as recently as six months ago was spacey verging on vaporous, is now emerging as the next must-have by enterprise IT groups. Bottom line: Web services got sexy and it all happened in late October.

Author Bio:

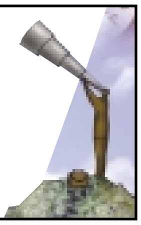
Robert McGarvey has covered the Web since 1994 for magazines ranging from *Technology Review* to *Upside*. He is the author of the best-selling book *How To DotCom* and a contributing writer for several SYS-CON publications, including ***Wireless Business & Technology*** and ***Web Services Journal***.

MCGARVEY@SYS-CON.COM

Hold on, because lots more happened in that same week. The Redmond Goliath weighed in, for instance, with an October 23rd announcement laying out its view of XML architecture for Web services – at a sprawling developers conference in Los Angeles, where chairman Bill Gates dramatically demonstrated Microsoft's .NET, the Web services platform Microsoft is betting on heavily. But the bigger point is that, unmistakably, Microsoft sees Web services as a central part of computing's future. Meantime, in the very same week, Sun Microsystems and its CEO, Scott McNealy, gave developers a look at the Sun ONE Web services strategy, and this wasn't just an opportunity for McNealy to take jabs at his .NET nemesis (although of course there was plenty of that, too). The Sun leader unveiled new and significant tools, including the Sun ONE Starter Kit, a package that promises to give developers what they need to start doing meaningful work in Web services.

"THE AMAZING FACT: all that happened in the same few days – the Web Services Edge West conference, Microsoft's PDC, Sun's unveiling its Web services strategy – and when all that activity is added up, there's no doubt. "Web services is at an inflection point," proclaims Nathaniel Palmer, Delphi Group Chief Analyst."

Web services have gone mainstream as the next *must-have* ***by enterprise IT groups — and it all happened in one*** *week in October.*



The amazing fact: all that happened in the same few days – the Web Services Edge 2001 West conference, Microsoft's PDC, Sun's unveiling its Web services strategy – and when all that activity is added up, there's no doubt. "Web services is at an inflection point," proclaims Nathaniel Palmer, Delphi Group Chief Analyst.

Web services, adds Steve Chazin, director of marketing for Bowstreet, an enterprise software developer, has become today's IT cover story and "2002 is poised to become 'The Year of Web Services.'"

Lico Talamantes, a senior consultant with SEI Information Technology, puts it even more directly: "Web services technology is ready to rock the marketplace."

More proof is that, early in November 2001, a parade of IT heavyweights lined up to issue their own Web services announcements. SAP, for instance, debuted mySAP; BEA announced Liquid Data; Borland announced new Java-based tools; Oracle unveiled Oracle9*i* Jdeveloper; and IONA unveiled Orbix E2A, a platform built from the ground up with Web services in mind and which,according to IONA, bridge the .NET – J2EE divide.

Whew, it's been hectic, but know this: "In the last 45 days we've seen a major shift in customer interest," says Steve Benfield, CTO of SilverStream Software and a keynote speaker at Web Services Edge 2001 West. A message he hammers home: "Customers now believe that Web services can help solve their problems. Web services," he adds, "will change the world."

Just that is a catch-phrase that emerged from the conference, and while Web services leaders may – and do – debate key points about how to deploy Web services, there's an emerging consensus that customers will line up to put in orders because they've now realized that Web services is the fast-track way to economically solve key problems.

"Companies – potential customers – couldn't quite get their arms around Web services, but that definitely is changing," observes Eileen Richardson, CEO of Infravio, a Web services developer, and a conference panelist. "Customers," says Richardson, "are seeing in Web services a route to increased productivity coupled with lower costs."

"What customers can expect from Web services, even today, is a real solution that can lower the cost of integration," adds Victoria Schmidt, a product marketer with IONA.

"Customers have spent an incredible amount on IT in the last decade. They want to leverage their investments to achieve new values," says Peter Graf, vice president, marketing, for SAP. "They want to find ways to use their IT infrastructure to reach outside the enterprise. That's why Web services is so significant. Web services lets you keep the systems you've invested in but use them to drive new values."

Why? How? Answer those questions by stepping back to the key themes put forth at Web Services Edge West and a big one is this: "Web services is the thing that lets the last three big new things work," says David Chappell, vice president and chief technology evangelist at Sonic Software and another conference keynote speaker, citing Daryl Plummer of the Gartner Group. He adds that a key factor in driving enthusiasm about the future of Web services is that never before has there been so much agreement in the IT

News from Web Services Edge 2001 West International Conference & Expo

WEST Global Unveils World's First Web Services Manager

(Santa Clara, CA & Dublin, Ireland) – WEST Global Limited, a provider of leading-edge Web services technology, announced the launch of its mScape Web Services Manager at the Web Services Edge 2001 Conference and Expo. mScape is the world's first Web services manager that enables companies to successfully deploy and manage Web services in production-grade environments.
www.westglobal.com

DataMirror Transformation Server Offers Real-time Database to XML Integration

(Santa Clara, CA) – DataMirror Corporation, a leading provider of enterprise application integration, has announced Transformation Server for XML, the latest addition to its award-winning Transformation Server software. Transformation Server for XML enables companies to capture, transform and flow data in real-time to and from common database formats and XML, the language of the Internet.
www.datamirror.com

Sonic Software Introduces SonicXQ

(Santa Clara, CA) – Sonic Software Corporation introduced SonicXQ, the reliable platform for Web services, at SYS-CON's Web Services Edge 2001 Conference. It provides a foundation for building and deploying distributed services across the extended enterprise with maximum reliability, security and scalability. SonicXQ supports Remote Procedure Call (RPC) and document exchange from a unified platform.
www.sonicsoftware.com

community. "For the first time in our lives, every major player in IT is behind one initiative," and with that much computing and marketing muscle behind Web services, there's ample reason for everybody – developers and customers alike – to stay optimistic, says Chappell.

A related fact: "When major vendors collaborated to push Web services standards, the industry began to notice," said Ben Brauer, product marketing manager for Hewlett-Packard's Web Services Operation.

A second crucial idea: "XML is based on free and open standards," said Charles Goldfarb, the father of XML, who delivered the opening keynote at the conference. What that means is simple: by using XML, various enterprise apps will be able to easily talk with each other, inside and outside the enterprise. Can they now? Not quite. Bits of enabling technology need to be standardized, but don't fasten on that because another big message from the conference is this: "If you are focusing only on technical details, you are missing the point," says Sean Rhody, editor-in-chief of ***Web Services Journal*** and a conference panel moderator. Exciting as the technology may be, and certainly there remain details to be worked out, this is something of a work in progress. But no matter, insists Rhody, because, more importantly, "Web services is a paradigm shift. With Web services, companies can do new, very interesting things that transform the enterprise."

Audience quality was rated outstanding by exhibitors

Dissonance

Is all copacetic with Web services proponents? Don't think that because – as vividly demonstrated at the conference and also by the dueling Sun and Microsoft announcements in the same week – there remain large points of disagreement. Is this cause for fretting that the Web services bubble is about to burst? Don't believe that, not at all. The disagreements, in fact, underline the tumultuous vibrancy that characterizes a community that recognizes it is center stage in IT.

Just where do proponents disagree? Conference attendees heard many points of

News from Web Services Edge 2001 West International Conference & Expo

SilverStream eXtend Provides Rapid Development With Oracle9*i* App Server

(Billerica, MA) – SilverStream Software, Inc., has announced that two products within the SilverStream eXtend product family, SilverStream eXtend Workbench and jBroker Web, have been tested and supported for deployment on the Oracle9*i* Application Server (Oracle9*i*AS). As a result, customers can now also leverage SilverStream products to rapidly develop and deploy J2EE and Web services–based applications on Oracle9*i*AS.
www.silverstream.com

Momentum Software Announces Major Web Services Initiative

(Austin, TX) – Momentum Software, Inc., a privately held e-business software development and integration consulting firm recently introduced the members of its new Web services team.

Web services are applications that save money and shorten development time by using a universal language to send data and instructions to one another. Scott Campbell, a member of Momentum's founding executive team, will direct the Web services initiative and was named Momentum's vice president for e-Business integration and Web services. Momentum also appointed Greg Heidel the senior technical lead for Web services for the company.
www.momentumsoftware.com

Sonic Software Introduces Products for the Distributed Enterprise

(Santa Clara, CA) – Sonic Software Corporation announced new software products for enterprise application messaging at SYS-CON's Web Services Edge 2001 Conference in Santa Clara, CA. Sonic Software's comprehensive suite of messaging middleware provides customers with a solid foundation for developing and deploying distributed services across the extended enterprise. The company announced separately the introduction of SonicXQ, a reliable platform for Web services, and SonicMQ 4.0, a new version of its leading E-Business Messaging Server.
www.sonicsoftware.com

Primordial Releases Product to Manage Consumption of Web Services

(New York) – Web services consultancy Primordial has announced the release of WSBANG! 1.0, an SNMP-compliant Web services proxy designed to help IT organizations master the paradigm shift towards Web services.

WSBANG! (pronounced "WHIZ-bang") gives IT central control over Web services consumption policies, as well as providing significant value-adds. It plugs into corporate networks and provides caching, monitoring, metering, micropayment tracking, reporting, and security and encryption. It allows IT managers to set Web services policy centrally and enforce it consistently across the enterprise.
www.primordial.com

Systinet Launched by NetBeans Founder

(Cambridge, MA) – Systinet (formerly Idoox), a leading provider of Web services infrastructure software, has announced the successful completion of its first round of financing. The investment will be used to extend the developemnt of the company's Web services platform and expand sales and marketing efforts.

Systinet was founded by Roman Stanek, the creator of NetBeans, which was sold to Sun

CEO Panel discusses the reality of Web Services

dispute and concern.

- ***Where are the customers hiding?*** One overarching fact is that as much as there is buzz now about Web services, firm orders are still lagging. But hold on since, predicts Rhody, that will change, and fast, even in the current torpid IT marketplace. “Within six to twelve months, this market will mature,” he predicts.
- ***All new, or old wine in a new bottle?*** A key point of debate: Are Web services a new, new thing? Proponents of the viewpoint that Web services is revolutionary include Annrai O’Toole, executive chairman of Cape Clear Software, and IONA CEO Barry Morris. Or is Web services an evolutionary extension of proven IT capabilities – a viewpoint firmly held by David Chappell, for instance: “Web services has evolved out of existing technologies. SOAP has been around since the mid-90s.” Guess what? Just maybe this fight, one of the liveliest at the conference, will be mainly of interest to IT historians because in the trenches, where customers are making decisions, Web services look to be a wholly new way of enabling applications to work together in ways that previously seemed unimaginable. Is that revolution or evolution? Probably the customer doesn’t care, not if Web services deliver on the promise of easier, low-cost integration.
- ***Will the IT power structure change?*** Another hot button at the conference: Barry Morris’s assertion that, with Web services, “you’re going to see the power structure of our industry change.” While Morris sees the new platform as opening tremendous opportunities for players that are comparatively small today, other conference attendees – while seeing potential for smaller players to earn big profits with Web services – nonetheless expressed belief that today’s IT leaders, from BEA to SAP, Oracle, and IBM, likely will continue to reap the greatest rewards as they aggressively move into Web services. Who’s right? Time will tell, but perhaps the key take-away thought is that huge profits will be earned by companies that articulate an early but coherent strategy for helping enterprises put Web services to work. This isn’t just about .NET vs J2EE (and as Annrai O’Toole told conference attendees, “Everyone is bored with Java vs .NET.”).
- ***Are Web services overpromised?*** Even some of Web services’ biggest fans openly wonder if the technology’s capabilities are

News from Web Services Edge 2001 West International Conference & Expo

Microsystems. According to Stanek, “We provide standards-based solutions for IT organizations, ISVs, and software developers to fully exploit the promise of Web services.”
www.systinet.com

Sonic Software Introduces SonicMQ 4.0 Enterprise Messaging Software

(Santa Clara, CA) —Sonic Software Corporation has introduced SonicMQ 4.0, the next generation of its award-winning messaging software for the reliable, scalable and secure transport of business-critical information throughout the extended enterprise. The announcement was made at SYS-CON's Web Services Edge 2001 conference. By providing greater breadth to its messaging solutions, increased interoperability with existing applications and emerging architectures, support for wireless messaging and increased industrial-strength capabilities, SonicMQ 4.0 ensures Sonic Software's leadership in the e-business messaging middleware market.
www.sonicsoftware.com

Infravio Ships World's First Web Services Management System

(Santa Clara, CA) – Infravio, Inc., a provider of enterprise software for developing, managing and deploying Web Services, is shipping the Infravio Web Services Management System (WSMS). Infravio introduced its application at the XMLEdge/Web Services Edge 2001 Conference & Expo.

The Infravio Web Services Management System is the only product in the Web services market that allows enterprises to become XML— and Web Services—enabled without adopting new development tools or discarding previous investments in enterprise applications.
www.infravio.com

New Standard for Scripting Languages and Web Services

(Santa Clara, CA.) – ActiveState and members of the open source community announced the Simple Web Service API at the Web Services Edge 2001 Conference & Expo. This API is a standard method for scripting languages to access Web services described with the Web Services Description Language (WSDL). For the first time, leading developers from the Perl, PHP, Python, Ruby, and Tcl communities are working together to create a common solution. With ActiveState coordinating their activities, a consistent, high quality implementation will be available sooner to the millions of programmers that use these languages.
www.activestate.com

Sun to Advance Web Services Plan

(Santa Clara, CA) - Sun Microsystems further defined its vision for Web services as it vies with rival Microsoft for control over the future of business computing on the Internet. Chief Operating Officer Ed Zander and other Sun executives recently advanced Sun's support for the standards behind the creation of Web services software. In addition, as reported by CNET News.com, Sun will begin selling a new instant messaging product aimed at corporate users, the iPlanet Instant Collaboration Pack.
www.sun.com

OASIS Forms Committee to Develop Web Services Component Model

(Boston, MA) – OASIS, the XML interoperability consortium, has formed the Web Services Component Model (WSCM) Technical Comittee to create a Web overpromised: "Web services are expected to be integration nirvana," says Hitesh Seth, chief technology evangelist at Silverline Technologies, an e-business consulting firm. That, as Seth points out, probably is not realistic. Web services will bring useful results but this is no silver bullet, say those who want the industry to adopt a go-slower stance. Others are more optimistic; a case in point is Benfield, who readily admits that Web services once were over-hyped, "but companies now are finding ways to deploy Web services that produce real results and the reality is that Web services let one machine talk to other machines. That's important." The take-away thought of conference attendees here is simple: promise only what Web services already can deliver because that will be plenty to stoke interest on the part of the large enterprises that are the first-wave target customers.

- ***B2B on steroids?*** Call this the most closely watched point of contention. Will Web services allow enterprises to easily use commonplace tools (XML and UDDI for service lookup) to find and expand B2B relationships – that is, will Web services help companies to effortlessly find new suppliers and even customers? Some Web services advocates think this is the next wave; others are more skeptical. Either way, Benfield, an ardent advocate of the benefits of the UDDI lookup capabilities, concedes that first deployments likely will be "inside the organization" – that is, Web services will let companies soup up their intranets. But as those test-beds show powerful results, watch out, suggests Benfield. A revived B2B marketplace may be a byproduct of today's Web services technologies.
- ***Just how easy is easy?*** A last, intriguing point of debate: How easy will and should Web services get? "Are Web services tools aimed at developers, or is the analogy to HTML, where the tools got so easy, everybody can use them?" asks Rhody. Just that is a central line of argument. Everybody at the conference agreed that, for the most part, existing tool sets remain too complicated for most users and, the next wave of tools will bring dramatically improved usability. But how usable should tools get? For now, leave that as an open question because the one fact is that Web services' future is still being defined.

Steve Benfield, CTO, SilverStream Software, and conference tech chair

And Finally...

Add it up, however, and even with the debates – maybe even because there remain points of contention amidst wide agreement about foundation points that enable Web services – nobody left the October conference thinking anything but upbeat thoughts. "This is simple technology, but it will change how business does business," says Benfield, and massive as his prediction is, conference attendees could only applaud his viewpoint that this is the IT solution customers have been waiting for.

Now, don't you wish you had been at the Web Services Edge 2001 West/XML Edge 2001 conference? But even non-attendees can take heart in the good news offered by Rhody: "Web services will continue to grow – it solves real problems, for real customers, cost effectively. This industry is now just beginning."

News from Web Services Edge 2001 West International Conference & Expo

services standard for interactive application access. WSCM will provide a coordinated set of XML vocabularies and Web services interfaces that allow companies to deliver Web applications to end users through a variety of channels. With WSCM, companies will be able to dynamically share Web services without the time and labor of creating multiple vendor-specific connectors in different Web languages.
www.oasis-open.org

PolarLake Previews PolarLake v1.2 at XMLEdge 2001

(Santa Clara, CA) - PolarLake recently previewed the latest version of PolarLake, its unique XML development and deployment platform for Java at the XMLEdge/Web Services Edge 2001 Conference & Expo. PolarLake version 1.2 adds additional Web services client and server support, including automated creation and deployment of Web services from existing COM and Java objects as well as the creation of new Java-based Web services from scratch.
www.polarlake.com

Industry Leaders Speak at Web Services Edge 2001 West

(Santa Clara, CA) – At the recent Web Services Edge 2001 West Conference & Expo, a number of industry leaders spoke and led conference sessions.

Dave Chappell, Sonic Software's VP and chief technology evangelist, delivered the keynote, "Web Services Meets Reliability."

Thomas Kurian, vice president of product development for Oracle9*i* at Oracle Corp. delivered a keynote on the evolving role of Web services in e-business and the importance of unifying Web services and J2EE.

During his keynote, Annrai O'Toole, executive chairman of Cape Clear Software, outlined how Web services are encouraging a welcome shift in the competitive strategies being used in the computer industry.

The Web Services CEO panel included Annrai O'Toole, executive chairman of Cape Clear; Dirk Slama, CEO of Shinka Technologies; Greg O'Connor, president of Sonic Software; Ali Kutay, former CEO of WebLogic and president and CEO of AltoWeb; Eileen Richardson, president and CEO of Infravio; and Barry Morris, CEO of IONA Technologies.

Steve Benfield, chief technology officer of SilverStream Software and Conference Tech Chair, delivered a keynote on "Unleashing Productivity: Web Services & Emerging Application Development Techniques."

Resonate CTO Brad Stone spoke on "Maximizing Enterprise Service Levels" and explored how service-level management solutions enable companies to view and effectively manage their e-business services.

Robert MacNeill, VP of SpiritSoft, led a conference session on how Web services can integrate with existing enterprise messaging systems, and how messaging can provide additional capabilities, such as network-based client caching and qualities of service.

Steve Wilkes, principal technologist at AltoWeb, led a conference session entitled, "The Challenges of J2EE," and discussed the benefits provided by J2EE and the challenges it presents when building enterprise applications.

Special .NET Focus Issue

Coming this February.
The most comprehensive .NET coverage!

WebServices JOURNAL .NET J2EE XML

The definitive issue on .NET

.NET: Wading Through the Hype to the Facts
by Sean Rhody, editor-in-chief of *Web Services Journal*

Bringing Buyers and Sellers Together: Real-World Web Services with Microsoft Technology
Is anyone we know doing it? Don't we have to wait for .NET or the next version of J2EE? Is it secure enough to be trusted?
by Eric Lynn

Application Networks: Unlocking the Value of Enterprise Web Services
Today, enterprise Web services require a higher level of reliability, security, and interactivity for any-to-any application communications – from back-end systems behind the firewall all the way to desktops, PDAs, cell phones, or any other device over the Internet. How do we provide it?
by Bob Pasker

Who's on the Other End of My Web Services?
In the paradigm of .NET, servers broadcast their available Web services utilizing UDDI. However, some services may contain varying levels of use depending on the authentication or security level of the identified user. What happens when aspects of authentication or the service itself fail?
by Keith Landers

Invoking .NET Web Services from Mobile Devices
Accessing Web services from a variety of devices
by Derek Ferguson

Microsoft Web Services Development Toolkit
A look at the new features available in this hot technology.
by Hitesh Seth

Another View: J2EE vs .NET
by Scott Dietzen, CTO of BEA

and more...

SPECIAL: .NET FOCUS ISSUE

WebServices JOURNAL .NET J2EE XML

WSJ2.COM

.NET: Wading through the Hype to the Facts

.NET Feature: Leveraging Web Services

Focus Discovery: UDDI and WSDL: Natural Companions

Technology: Exploding Web Services Myths

Focus Discovery: Second-Generation Portals

Focus Discovery: Web Services & Distributed Objects Computing are Complementary

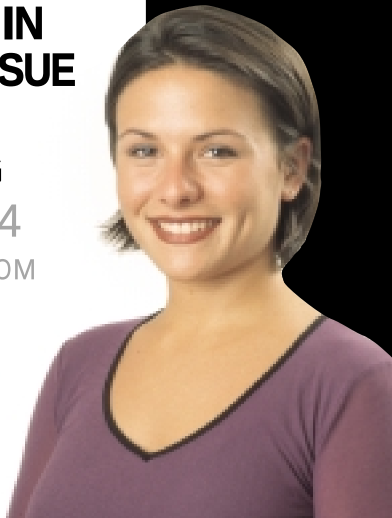
.NET Feature: Setting the Standards

Focus Discovery: An Introduction to WSDL

Feature: Web Services & Distributed Component Platforms Part 2

SUBSCRIBE ONLINE and SAVE
www.sys-con.com

TO ADVERTISE IN THIS SPECIAL ISSUE
CONTACT
MEGAN RING
(201) 802-3024
MEGAN@SYS-CON.COM



SYS-CON MEDIA

Don't miss this special issue!

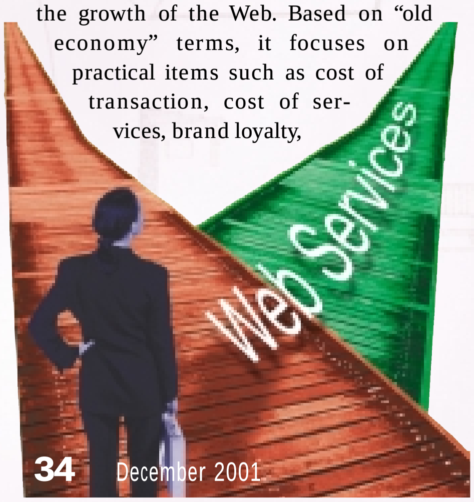
Written by Hal Stern

Second-Generation Portals

Poised to deliver second generation Web services

During the rise of the Internet, the portal's roles as an aggregator of content, a means of personalizing Web access, and a conduit to various online communities became a mainstay of life on the Web. Accordingly, portal sites enjoyed the position of "sure thing" players because they delivered value in traditional media terms: an audience that was easily measured in terms of hits, page views, and memberships. But today, "portal" has become an overloaded term, implying everything from network access to the familiar stock quotes, news, and weather aggregation services.

However, a second generation of portals will soon bring a new focus to the portal concept, one with enduring value and a solid economic foundation. In the wake of the great dotcom implosion, a new set of metrics is governing the growth of the Web. Based on "old economy" terms, it focuses on practical items such as cost of transaction, cost of services, brand loyalty, and customer satisfaction. The next generation of portals will become the prime delivery vehicle for a new set of network services that satisfy these metrics. These new services are generically termed Web services, and will become the dominant technical and business model for the next generation of networked applications, including those that drive Web sites.



In the broadest sense, Web services refers to networked access to highly focused application functions, using XML messages in a loosely coupled fashion. Web services include a number of attributes that offer a vastly expanded range of possibilities compared to the present model based on tightly coupled interaction inside the firewall. Web services are highly flexible in terms of connections, offer deep granularity in application functions (as opposed to today's monolithic packages), and fully support the construction of systems that become a "network of networks." Taken together, these attributes support the genesis of second-generation portals with significantly expanded capabilities that will make them the prime delivery vehicle for the coming wave of Web services.

The move from relatively isolated, large-scale applications to finer-grained Web services will have two major impacts on portal servers. First, Web services require a multiplicity of portal delivery channels to any number of devices, ranging from PCs to LCD displays in automobiles. Second, careful architecture and application design with portal delivery channels in mind will create "services on demand" which offer significant benefits to both businesses and consumers, such as "anywhere, anytime" access.

Portals

The integration of Web services into portal environments will represent a major shift in how a portal is ultimately defined. The first generation of portals, which still dominates major Web sites across the Internet, can be thought of as a combination of Yellow Pages and a clipping service. As a Yellow Pages directory, it simply offers a convenient method of navigating to sites of common interest for the portal's user population, functioning as a switchboard to connect individual clients to a community of services. As a clipping service, it processes streams of data such as stock quotes or news headlines, and formats this influx into HTML for viewing on client browsers. Both of these functions are based on delivery of relatively shallow content – clicking on a news story merely displays the story but has no other relationship to applications within the enterprise. Portals haven't been used to foster user interaction in a way that substantially enhances the user's productivity, especially in commerce environments.

In the second generation of portals, what was once static becomes largely dynamic. The portal's data channels are no longer passive conduits for an influx of data. Instead, they become interfaces to programs performing active functions on behalf of the user. A content channel in a portal can be thought of as an active content generator, driven by user or external events. Designing applications for this content-oriented delivery mechanism redistributes the processing load in a way that allows data to be delivered in highly specific, value-added ways to the end user. The portal becomes the underlying application's API consumer, intermediating between users and their collection of applications so that aggregation or process-driven application coordination become the typical user work flows.

This new order allows the user to interact in a way that provides highly customized information. In effect, the portal is leveraging the API to create a dynamic content channel. Each channel is an object representing the application that feeds the channel. Portals can now be developed so that they become collections of such objects arranged

Author Bio
Hal Stern is the CTO of iPlanet. In that role, he identifies and leads adoption of distributed technologies, aligns iPlanet and Sun Microsystems engineering functions, and maps high-level vision into practical technology approaches. He was the first employee from Sun's field organization to be promoted to Distinguished Systems Engineer. Stern is the coauthor of *Blueprints for High Availability*.
HAL.STERN@SUN.COM

to maximize the user experience.

To provide these dynamic content channels, second-generation portals encourage the shift of processing to the server side and continue the trend toward "thin" clients. Rather than thick clients that perform editing, local logic functions, or presentation management, those functions are handled by the server-side business logic or the multi-channel delivery modes of the portal server itself. As a return on this investment in new architectures, software developers will have greatly expanded flexibility and control when developing transactional applications.

For example, consider an application that performs transactional updates of several systems and has been targeted at a wireless handset for consumer use. In a browser-centric world, or a desktop thick-client model, it's tempting to use a heavier-weight client or client-side applet to perform some of the transaction sequencing. Using a second-generation transactional portal, the work of ordering the transactions, handling failures and recovery, and providing updates to the user is handled entirely on the server-side. Factor in a lower reliability client-side environment such as a wireless handset, and the benefits of refactoring the business logic with a heavy emphasis on the server side are clear; the portal server provides device independence when the heavy lifting is safely on the well-managed, predictable side of the data center.

This new breed of portal provides an ideal solution in the current e-commerce milieu, where the demand for portal technology is firmly established. Although portals first gained a foothold in the business-to-consumer space, where they were the province of large-scale ISPs, there's now a clear recognition that portal technology will play a critical role in enterprise infrastructures, where they'll help employees make sense of an overwhelming amount of data from both within and without the corporate firewall.

This compelling problem of data overload has already caused the market for Virtual Private Networks to grow to more than $1 billion in 2001 as enterprises attempt to develop expensive, customized methods of giving employees "anywhere, anytime" access to corporate information.

In contrast, second-generation portals will give enterprises an off-the-shelf-approach to this problem that promises to be highly cost-effective. A portal server – and its surrounding infrastructure – will provide a highly flexible template that can be quickly adapted to specific corporate needs. A host of industry standards, such as the J2EE platform, XML, SOAP, and many others, help to make this template universal in nature. Thus, the primary growth market for transactional portals is likely to be within the enterprise, where cost savings and improved decision-making are enabled by providing employees, partners, and customers with direct access to the right combinations of applications.

New Flexibility

As transactional portals grow in number and scope, they enable further flexibility in the number and types of applications developed, and expand the scope of interactions these new applications have with existing internal and external systems. Logical connections between systems will be driven more and more by user demands, as opposed to the dictates of tightly coupled applications. This growing diversity of highly specialized applications and the density of their interactions will require a new form of control and coordination that shields the user from all the underlying complexity it produces. The solution will be a highly evolved form of Web services called services-on-demand, which continuously monitor the user's current situation in terms of device currently in use, geographical location, and notification preferences.

"The next wave of growth on the Internet will be fueled by business models that rely on decreased cost or new sources of revenue"

This new type of Web service remains continually aware of all aspects of the user's present context, both in a formal and personal way. For instance, a presence-enabled service knows who the user is, his or her present role and preferences, privacy requirements and security level, past history in terms of required service, adherence to service agreements, geographical location, and type of device currently in use. In more mundane terms, the service recognizes that if you're in a meeting with your cell phone turned off, an SMS message won't be nearly as effective as a text page, but both are preferable to an instant message if you're not online at the time.

How will these services-on-demand be integrated into second-generation portals? To recap, the dominant feature of the new portals themselves will be the inclusion of dynamic data channels that coordinate activity between user applications. To make these channels "smart" in the sense described above will require an entire architecture where the portal presentation aspect is coupled with Web and application servers, as well as functional blocks driving policy, network identity, and service management. Also required will be provisions for automated workflow that couples with back-end databases and legacy systems, in addition to other Web services.

The dynamics of this architecture are best understood by following the flow of data on one of the portal's dynamic channels. When a service request comes in over the portal, it includes the user's present context in terms of location, current access device, role, and even time of day. This information is captured and fed into the application server, the central engine where the Web service executes. Here the service calls upon other sections of the architecture to examine and process parameters such as current context, policies regarding the present request, and applicable business rules. This includes orchestrating the service's overall workflow and coordinating tasks such as accessing databases, legacy apps, and other services. It also includes consulting an entire range of policy microservices covering items such as identity, security, privacy agreements, and specific user preferences and history. At the same time, another series of microservices cover management tasks, such as billing and enforcing quality of service levels. When the service is prepared to output a response, it uses the portal to generate the response in the proper context, such as reformatting for wireless interfaces and personalizing it for a specific user.

The next wave of growth on the Internet will be fueled by business models that rely on decreased cost or new sources of revenue; services-on-demand will be a major driver of that growth because the fundamental architectural goals are to connect applications and data assets to the right communities of interest at the right time. With this type of architecture in place, it becomes clear that the next generation of portals is poised to deliver a whole new range of services-on-demand capable of adding significant value to those businesses and individuals that use them.

Reviewed by Joseph A. Mitchko

About the Author:
Joe Mitchko is a senior consultant with Phase II, a leading consulting firm specializing in Internet and database-related architecture based in New Jersey.
JMITCHKO@RCN.COM.

WebLogic Server 6.1

by BEA Systems

Leading application server gets Web service interface lift

BEA SYSTEMS, INC.
2315 North First Street
San Jose, CA 95131
Tel: 1 800 817-4232

FOR SALES INFORMATION:
Web: www.bea.com
E-mail: sales@bea.com

TEST ENVIRONMENT
Dell Inspiron 3800, 512M RAM, 18G H/D, Windows 2000 Professional (Service Pack 1)

A guiding principle in the software business is that everything designed and developed today will eventually become a legacy system tomorrow. Just yesterday I was all excited about the new world of client/server. For a time, it was client/server this and client/server that, accompanied by a plethora of buzzwords to fill the minds and résumés of professionals. Almost overnight, however, Internet application architecture involving Web browsers, Java application servers, etc. took over the scene. Client/server is now a legacy technology.

With this idea in mind I have come to wrestle with a new thought: distributed components, J2EE architecture, and others will eventually be considered legacy systems as well.

As I was looking over the new Web service capabilities packaged in BEA's 6.1 release of its WebLogic server, I realized the playing field was shifting again, especially since the major vendors are beginning to seriously move ahead in support of Web services. BEA's approach to implementing SOAP-based Web services moves us even closer.

BEA announced the general availability of WebLogic 6.1 on August 1, 2001. BEA has made substantial progress in support of Remote Procedural Call (RPC) and message-based service requests using the Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) 1.1 specifications. Since these specifications only define the messaging protocol, vendors are still free to be creative in how they implement the interfaces. The approach taken by BEA is innovative, yet, depending on the complexity of your XML data structures, it may have a few drawbacks.

Overview

WebLogic 6.1 classifies a Web service as either RPC-based or message-based, depending on whether the service transaction is synchronous and requires an immediate response or is asynchronous and doesn't require an immediate response. All SOAP functionality in WebLogic is based on implementing one or the other. Now equate this class of service to an EJB component.

RPC services are handled by a stateless session bean and message-based services are handled by a message-driven session bean. This is how BEA implements SOAP-based Web services in its WebLogic server.

Once you have the underlying EJB component designed, coded, and tested, all you need to do is set up an XML-based build descriptor file to get your service up and running. WebLogic will automatically generate the SOAP interface and compile the WSDL description for you based on the EJB design. BEA has also done a nice job of bringing SOAP services into the J2EE fold.

You may be asking yourself what happens if the "legacy" application I am wrapping in a SOAP service is not based on EJB technology? The answer to that is that you'll still need to develop an EJB session and/or message bean to serve as an adapter to the legacy application.

Getting Ready

Here are a few things to keep in mind before developing a Web service in WebLogic. The 6.1 version of WebLogic is fully compliant with the SOAP 1.1 standard (plus attachments) and the 1.1 version of the WSDL document, although, according to BEA, Web services currently ignore the actual attachment of a "SOAP with Attachments" message.

Moreover, you're not required to have extensive knowledge of SOAP and WSDL to develop a Web service. As I alluded to earlier, the WebLogic 6.1 Web service interface is tightly coupled to the remote interface of a matching EJB component. As a consequence, if your application is not EJB-based, you'll need to develop an EJB component wrapper for it.

WebLogic provides all the necessary build tools to transform and deploy your EJB interface into an ebXML/SOAP-based interface. You can assemble your own deployment jar if the standard build does not fully handle what you have designed. You'll find the process well documented, with plenty of details to guide you in rolling your own deployment .jar file.

The build process pretty much does all the work, configuring and integrating the various libraries to interpret the SOAP request and reply messages, and generating the appropriate WSDL document describing the service. Again, you must know your way around EJB components to deploy a SOAP service.

Creating a Service

To begin developing your service, you'll need to analyze the characteristics of the application to which you are binding your service. Is the service best implemented using an RPC type of interface, or should it be message-based? The documentation provides some useful questions to guide you in the right decision. Next, once you have an idea of the class of service you're developing, you need to develop the appropriate interfaces using session or message-driven EJB components. If the application is using EJB components on a WLS platform, you can build upon already existing stateless session beans.

In the case of a message-driven, asynchronous Web service interface, you'll need to develop a message-driven bean and set up the various JMS messaging queues on the server. Explaining how to develop both types of beans in WebLogic is beyond the scope of this review. WebLogic provides you with some wonderful EJB code examples and plenty of documentation – perfect for self-learning.

The next step in developing a Web service is to deploy the EJB components on the WebLogic application server. This is primarily a compile and deployment process that can be done manually or automatically by a build process.

Next you'll need to write an XML-based file called *build.xml* that defines the service name and other "Ant" specific directives. This file, along with the EJB component .jar file, serve as inputs to the "WSGEN" build process.

The output of the "WSGEN" build is an Enterprise Archive file (.ear file) and contains everything necessary to run a Web service. Deploying the Web service is as easy as placing the file in the appropriate WebLogic directory.

The "WSGEN" build process was built on top of Apache's Jakarta Project Ant (see Jakarta.apache.org/ant), a Java- and XML-based build utility. Ant is embedded in the WLS library, so it's not necessary to separately download the utility.

Developing a Client

The implementation goes further than helping you develop a service; it actually makes it easy to develop a client, too. There is support for Java clients and (surprise) Microsoft Visual Basic applications.

The WSDL is used by the logic resident in the client.jar file. This file contains everything necessary (and nothing more), enabling a client to access the service. The SOAP and WSDL interpretation, data binding, and so on are handled by the client.jar library, and for the most part it's transparent.

There are two approaches to designing a Java client: static and dynamic. The static approach requires the remote interface to be instantiated on the client while the dynamic approach uses a Web service proxy object to make the calls to the service. A dynamic client doesn't require the remote interface.

The client API calls are somewhat similar to remote EJB client calls but much more streamlined. For example, you still need to obtain an initial context to call a service, but instead of calling an EJB factory for it you'll be accessing it from a SOAP connection factory. Moreover, static calls to the service are made through a remote interface. This provides a simple but proprietary approach to performing SOAP service request calls. In a way, it makes the SOAP implementation transparent for both client and service. You are free, of course, to make a SOAP request any way you choose, as long as you follow the SOAP and WSDL specifications.

Pros and Cons

The WSDL and the client.jar files are available from the same Web application as the SOAP interface servlet. All a developer needs to do is download the WSDL and client.jar files from the service's Web site, design the client (the WSDL file is interpreted), and include the client.jar file in the CLASSPATH. The client.jar file is designed with a small footprint, therefore, supporting a thin client. WebLogic also provides the necessary .dll files for Visual Basic–based clients. The functionality for both Java and VB is the same: converting SOAP requests.

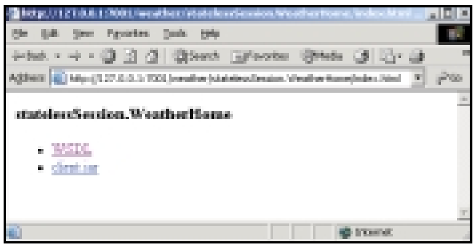


FIGURE 1 An example of a Web services home page

There are some limitations to WebLogic's approach in implementing a Web service. Complex XML data structures represented with XML schema can't easily be managed by the WLS 6.1 integration process. In other words, the binding between the SOAP interface and the legacy application is made strictly through an EJB interface, requiring complex structures to be represented through a one-dimensional Java bean structure. There are standards emerging in this area to help you navigate and bind the XML service parameters to the host application's interface points, but they are much more difficult to code compared to having your XML to Java bindings done automatically for you.

WebLogic 6.1 doesn't provide any testing utilities or automated tools. However, given that the SOAP interface is generated automatically by the WebLogic WSGEN process, automated regression testing of the SOAP service could instead be implemented at the EJB component level. This may make sense since there are more testing tools out there for EJB components than there are for SOAP-based Web service interfaces. Of course, any change to the EJB interface (and subsequently to the SOAP interface) would be flagged as an error. You may still want to extensively test the SOAP interface using a test client to make sure that the fundamental binding from EJB to SOAP request is correct.

Obtaining a Copy

WLS 6.1 can be downloaded from the BEA Web site. It comes with a 30-day free license. Be prepared to download a 73-meg file.

A Sound Implementation

With the WebLogic approach, you're really designing a SOAP service through an EJB component. The SOAP service is, in a sense, an extension of a stateless session bean for RPC services and message-driven beans for asynchronous, extended transaction types of services. Whether this approach is good or bad depends on your implementation. At this time, it is difficult to compare it with other products because of the limited number of vendors supporting SOAP. Moreover, the SOAP 1.1 standard doesn't specify how the interface is to be implemented, so we can't use that as a gauge either.

The BEA approach poses other questions as well. Is it easier to design a Web service starting with the SOAP interface and working through the integration issues from there? Or is it easier to design the EJB component, test it, and then have the service automatically generate the SOAP and WSDL XML? There are more EJB development and test tools out there, so deploying SOAP services as transparent extensions to an EJB component layer would, in my opinion, speed up the development process.

One particular downside to passing SOAP requests through the EJB interface is that it forces you to represent complex XML data types using a class structure (typically consisting of Java beans). This makes it awkward to use complex XML structures for parameter and return values, requiring you to implement a class representation that is mapped to the complex XML type during the WSGEN build process. Therefore, you may find it a little challenging to implement a WebLogic SOAP service that needs to dynamically navigate through a complex XML parameter and to be able to arbitrarily bind program variables to elements.

Overall, BEA's implementation is sound and the approach does make it easier for the average developer (yours truly included) to implement a SOAP service. As BEA and other leading vendors begin to adopt and implement the various Web service specifications, I can envision SOAP-based Web services bubbling up all over the Web in the not-too-distant future.

Effective Web Services

WRITTEN BY

Charles Stack

Charles Stack is founder, president. and CEO of Flashline, the leading provider of enterprise software reuse solutions. Stack has been managing software development for over 20 years, with nearly a decade of experience managing the development of online systems. He is credited with founding the first Internet retail store and is the inventor of two patented ecommerce applications.

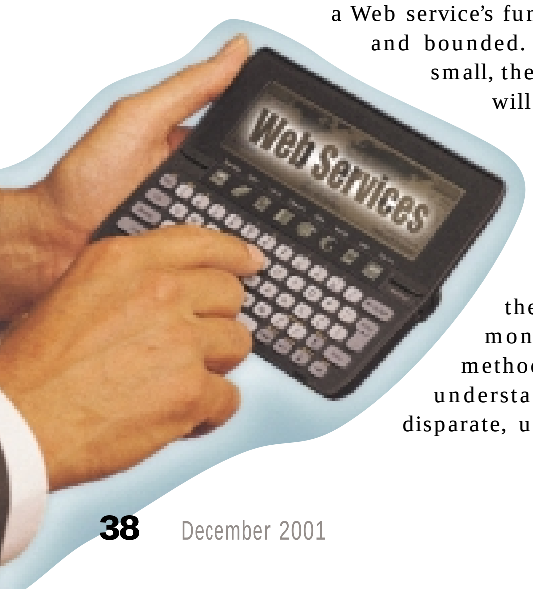
EDITOR@FLASHLINE.COM

Reusable software components – when properly built, promoted, and tracked – deliver an enormously productive alternative to traditional "built-from-scratch" application development. The benefits are real, tangible, and ultimately reflected in the bottom line for those organizations with the wisdom to recognize software reuse as an adaptation of the same concept that made Henry Ford famous and very, very rich. Web services, when held to the same standards of construction, promotion, and tracking, offer the same benefits. The advantages of component-based development (CBD) apply equally to service-based development (SBD) – better, faster, cheaper creation of software solutions.

The characteristic benefit of software components is their ability to encapsulate functionality within a public interface. Encapsulation allows systems of ever-increasing complexity to be efficiently built, managed, and maintained as a series of high-quality, loosely coupled, interacting parts. Web services exhibit the same characteristics. To be effective, a Web service should only expose or make public those methods necessary for its usage. Other methods or properties should be hidden within the service's "black box." The direct benefit of appropriate encapsulation is simplicity and ease of use.

The Web service interface and methods must be immutable. As with component reuse, you must not change method names after a service has been deployed. An implied contract exists for deployed services that requires constancy of interfaces. This requirement of immutability mandates that the component be extremely well-thought-out prior to deployment.

Other CBD issues that are equally applicable to Web services include granularity, modularity, cohesion, and coupling. In applying the lessons learned in CBD and reuse to the matter of Web services, we know that a Web service's functionality must be properly sized and bounded. If the service is too granular or small, the overhead of multiple connections will have a negative impact on its performance. Conversely, excessively large, monolithic services are confusing, difficult to implement, and unlikely to be reused across applications.



The relatedness or cohesion of the service's methods must also be monitored. Services should contain methods related to a common, clearly understandable goal. A multitude of disparate, unrelated methods reduces opportunities for reuse, confuses developers, and again makes the service difficult to manage. Like components, services should be loosely coupled and present the appearance of being entirely self-contained. The primary service may utilize other services to perform a given function, but the application should interact directly and exclusively with the primary Web service. It should never be the application's job to coordinate interaction between multiple services.

Web services must also be flexible — able to support current requirements and still be adaptable to future needs. Business needs are constantly changing, so the adaptability of software systems should always be a core functional requirement. The same adaptability that makes systems flexible within an application also enables reuse across applications. Services should therefore be designed to be customizable, portable, and generic.

One method for incorporating flexibility into services is to provide each service with a set of signatures. Each signature, represented by a variable passed to the service when called, would define a variant of the service's functionality. The signature would tell the service which modality the application was expecting the service to provide. The use of signatures also allows a service to utilize in-place upgrades (subject to standard predeployment testing, of course) where modified functionality can be added without interrupting the service or breaking existing applications.

Services that are intended for use beyond a single initial implementation require a high level of documentation. The documentation set for Web services should include comprehensive method descriptions, sample application code, complete response codes, tutorials, use cases, and architectural models.

Testing is another key piece of documentation required before Web services can be effectively deployed. Any application developer must know the level of reliability of a given service, as well as its specific response characteristics and methods.

Flashline has spent years building expertise in software reuse, which is reflected in the Flashline Component Manager, Enterprise Edition (CMEE). Much of what we have learned about reuse and CBD applies equally and just as effectively to Web services. While significant differences remain, the similarities between CBD and SBD are so striking that to describe the two as parallel methodologies may be inaccurate. Web services may in fact be the next step in the evolution of components.

We don't mind if they say we publish "geek" magazines, as long as we're known as...

The World's Leading *i*-Technology Publisher

June 2001
136,000 copies in print

June 2001 Preview
250,000 copies in print

June 2001
82,000 copies in print

June 2001
72,000 copies in print

June 2001
48,000 copies in print

Premiering December 2001
20,000 copies in print

Premiering January 2002
20,000 copies in print

Premiering February 2002
150,000 copies in print

and Producer of the World's Leading *i*-Technology Developer Conferences

JDJ EDGE conference&expo

XML EDGE conference&expo

web services EDGE conference&expo

COLDFUSION EDGE fast track

wireless EDGE conference&expo

Inc. 500 1999 WINNER

SYS-CON™ MEDIA SYS-CON™ EVENTS

www.SYS-CON.com

America's fastest-growing privately owned publishing company

Inc. 500 2001 WINNER

Written by Alan Conway

Web Services and Distributed Objects: Competing or Complementary?

Roles for both in total business integration

It's tempting for those with a background in distributed objects to look at the upcoming Web services standards as just another set of distributed object technologies, probably with some misgivings about reinventing the wheel. Conversely, with the advent of Web services it may be tempting to think that distributed objects are no longer important. After all, Web services have

- A transport protocol: SOAP (Simple Object Access Protocol)
- Something like an interface definition language: WSDL (Web Services Definition Language)
- A location service: UDDI (Universal Description, Discovery, and Integration)

However, in this article I'll explore how Web services complement rather than compete with traditional distributed object platforms like CORBA and J2EE. When used correctly, Web services and distributed objects have different paradigms: *document-oriented* and *object-oriented*. A combination of both paradigms is required to achieve end-to-end business integration.

The object-oriented platforms CORBA and J2EE have been tremendously successful for building and integrating applications within the enterprise. The document-oriented XML technologies associated with Web services are rapidly gaining popularity for business-to-business integration between enterprises. Total business integration needs to address both of these areas.

Object-Oriented vs Document-Oriented

To a distributed object veteran, a Web service looks like yet another distributed object system. Based on this comparison, it's tempting to simply expose existing CORBA or Java objects as Web services by mapping them to WSDL. However, XML has a different heritage. It's document-oriented, not object-oriented. Web services should be considered a medium for implementing document-exchange systems and not as simply a new flavor of distributed object systems.

How Document Exchange Is Different

In a document-oriented system, document design is key. Interfaces are just a way to pass documents. Contrast this with object-oriented design, where interfaces are key and data structures are just convenient packages of data for the operations on the interfaces.

Document-exchange systems have:

1. Larger, richer data structures: so much richer that we call them "documents"
2. Looser coupling between agents
3. More asynchronous communication

Richer content enables looser coupling, which in turn enables asynchronous communication.

Object-oriented systems behave like people having a conversation. document-exchange systems behave like people sending old-fashioned paper documents to each other.

Suppose I want to buy a hat. The OO approach is to find a hat shop and talk to a salesperson (object). A request/response session follows: "Do you have a size 16 black fedora?" "We don't have that color." "What color fedoras do you have in size 16?" "Brown and beige." "I'll take a brown size 16 fedora." "That'll be $19.95."

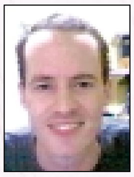
Things to note:

- Simple data structures: hat type, quantity, list of colors, price
- Tight coupling: Replies are meaningless without the context of the question. The phrase "brown and beige" tells you nothing useful on its own. Also, I have to understand who I'm talking to (the interface). I have different conversations with hat salespeople than with waiters
- Synchronous communication: Because of tight coupling, I must be able to associate replies with requests. My subsequent requests often depend on a previous reply

Now the document-exchange approach: I send a letter to a mail-order hat company requesting their catalog. A few days later, I receive it by mail. Reading the catalog tells me the hat is available and their prices. I fill out an order form and send it back with a check.

Things to note about the document exchange:

- ***Complex, structured, self-describing data***: I got an entire hat catalog. I knew it was a catalog by reading it, not by knowing that it was a response to my request.
- ***Loose coupling***: I didn't need to directly associate the catalog and my previous request because the catalog is sufficiently self-describing to anyone who knows how to read a catalog (understands the schema).
- ***Asynchronous com-***



Author Bio

Alan Conway is a senior engineer at IONA Technologies and has been with the company for more than five years. He has extensive experience with both CORBA and J2EE. Conway has a keen interest in the emerging Web services area and how it complements established infrastructure technologies.
ALAN.CONWAY@IONA.COM

munication: Enabled by the self-describing data and loose coupling. I couldn't know directly that the envelope containing the catalog was a response to my request, I only knew that indirectly by opening it up and reading the catalog inside.

Granularity of Service

Web services aren't objects in the classical OO sense of the word. They're coarser than typical distributed objects, so mapping distributed objects directly to a Web service will probably give you a poorly designed Web service.

That doesn't mean you shouldn't map distributed objects to Web services! On the contrary, distributed object platforms are the ideal underpinning for an enterprise-grade Web service. It does mean you need to design some of your distributed objects with the knowledge that they're going to be Web services. These service objects will typically delegate their work to many smaller-grained distributed objects in a distributed system underlying the Web service and will typically pass larger chunks of data in a single call than the objects in that system.

To a distributed object veteran there's a strong analogy here. Consider CORBA or EJB objects in relation to nondistributed C++ or Java objects. You can expose just about any Java or C++ object as a distributed object, but good distributed system design dictates that the distributed objects are more service oriented than the fine-grained, non-distributed objects that they use internally to implement their interfaces. There is a natural progression: Web services are built on distributed objects, distributed objects are built on nondistributed programming objects. At each step the granularity becomes finer.

Where Does Each Approach Work Best?

The short answer is to use distributed objects to build high-performance systems with a well-defined architecture, and Web services to connect independent systems in a loosely coupled fashion based on shared business processes and documents.

Distributed Objects

Distributed objects are an extension of classic OO programming techniques familiar to developers. Well-designed distributed object systems can deliver very high performance and scalability, as has been proven many times in live deployed systems. Classic examples are manufacturing process control and telecom switch management.

A distributed object system requires a well-defined architecture with stable interfaces between components and some level of system-wide agreement on infrastructure technology. This requirement can be a problem for business-to-business integration because there may be no agreement on infrastructure and no system architecture shared between trading partners. Often the most you can hope for is a set of agreed-on business processes and document formats.

Document Exchange

Document exchange is an electronic version of the traditional paper-based processes familiar to business people. It works best in the business world, where it can reflect traditional business documents and processes. Web services are the natural way to implement document exchange because they build on XML and HTTP. These standards currently are pretty much the only choices for portable document representation and Internet communication respectively.

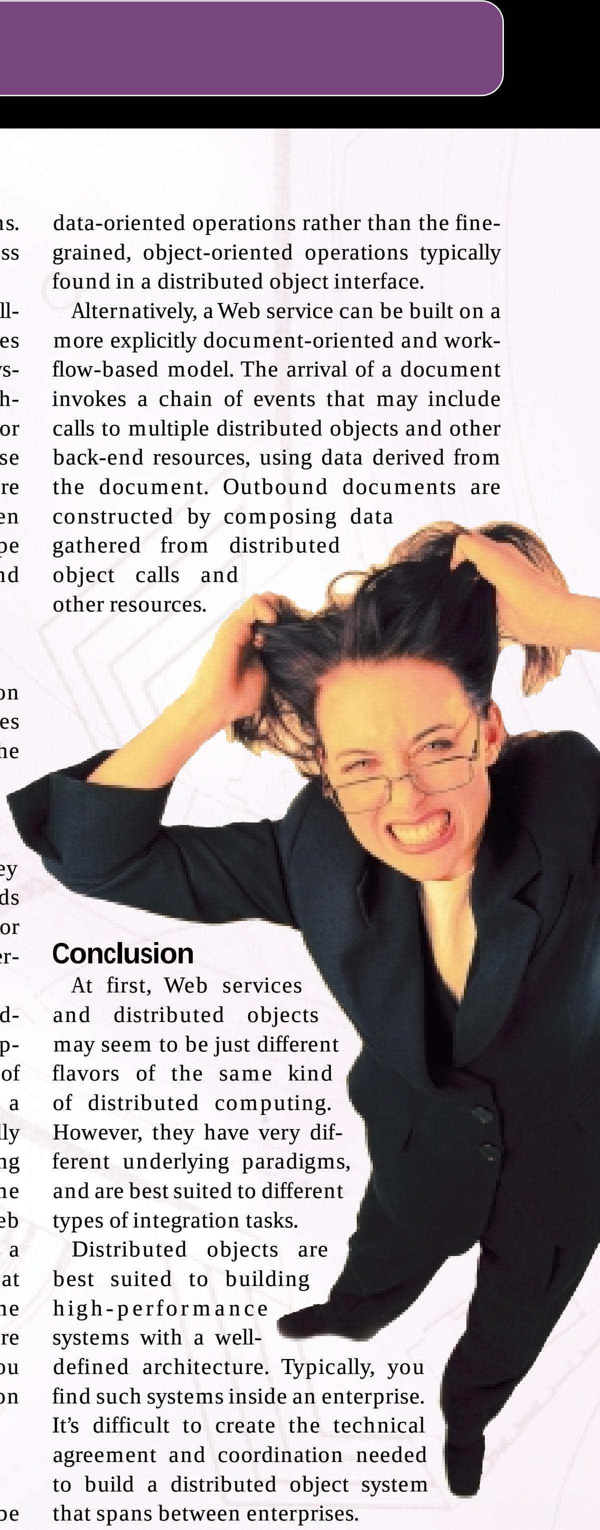
Web services are unlikely to perform adequately as high-performance system components due to the size and parsing overhead of XML documents. It's hard to imagine a telephone exchange being successfully implemented as a set of intercommunicating Web services. Yet it's easy to imagine some types of access to such a system as a Web service. For example, for regulatory reasons a telecom carrier might build a Web service that allows other carriers to request phone numbers on their network. Web services are ideal for this kind of interaction, where you can't assume that there'll be agreement on CORBA or EJB interfaces.

Bringing Them Together

Document-oriented Web services can be implemented on top of distributed object systems. Distributed objects are the building blocks of a robust, scalable, highly available service within an enterprise. Web services are highly accessible entry points to those systems. They enable automation of shared business processes between trading partners with very different internal systems.

A Web service can be built by directly mapping the interface of a distributed object. As we have mentioned, the interface should be designed to be document-oriented to make a good Web service. That means coarse-grained, data-oriented operations rather than the fine-grained, object-oriented operations typically found in a distributed object interface.

Alternatively, a Web service can be built on a more explicitly document-oriented and workflow-based model. The arrival of a document invokes a chain of events that may include calls to multiple distributed objects and other back-end resources, using data derived from the document. Outbound documents are constructed by composing data gathered from distributed object calls and other resources.



Conclusion

At first, Web services and distributed objects may seem to be just different flavors of the same kind of distributed computing. However, they have very different underlying paradigms, and are best suited to different types of integration tasks.

Distributed objects are best suited to building high-performance systems with a well-defined architecture. Typically, you find such systems inside an enterprise. It's difficult to create the technical agreement and coordination needed to build a distributed object system that spans between enterprises.

Web services are best suited for implementing shared business processes between enterprises. A Web service provides a common facade for enterprise-specific systems, making it easier to create the technical agreement needed for automated business-to-business integration.

The technologies work hand in hand: distributed objects provide the underpinning for the systems that run each enterprise; Web services provide the connectivity between the systems in independent enterprises.

.NET FEATURE

written by Hitesh Seth

Setting the Standards

Referred to by some as the fourth wave of computing, Web services intend to standardize the way business applications communicate with each other, whether they are within an enterprise or communicating with external partners, customers, and suppliers. In this article we explore the key standards behind Web services, as well as the various tools and technologies that the upcoming Microsoft .NET provides for developing and deploying dynamic Web services.

What is a Web Service?

I like MSDN's (Microsoft Developer Network) definition of a Web service. A Web service is a unit of application logic providing data and services to other applications. Applications access Web services via ubiquitous Web protocols and data formats, such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented.

Why Web Services?

In a typical large enterprise, many application delivery models and frameworks exist today. For instance, a number of departments may use the Microsoft ASP/COM-based frameworks, another group may utilize one of the powerful application server environments, and another group may have developed their own proprietary application framework. Of course, there is a need to standardize architectures and frameworks across an organization, but getting there is a challenging and in some ways impossible task. What is needed is a mechanism where a number of

Formalizing how business applications communicate

Author Bio:
Hitesh Seth is chief technology evangelist for Silverline Technologies, a global e-business and mobile solutions consulting and integration services firm. Hitesh has extensive experience in the technologies associated with Internet application development, including electronic commerce, content management, EAI, B2B Integration, Enterprise Information Portals and wireless/mobile application development.

HITESH.SETH@SILVERLINE.COM

systems, both internal and external, can be integrated in a loosely coupled fashion. Because Web services are based on open protocols and standards, the participating systems can be implemented in different technologies but still be available under a single coherent space. Change is also inevitable in corporate IT, where a popular application framework today will soon be outdated and a better framework will be available tomorrow. Web services can be effectively utilized in this scenario: while going through the migration a company may choose to expose a number of core features of their ASP/COM-based application using Web services and then slowly migrate their user interface, interaction, and eventually the complete application into the .NET flavor.

Web Services Standards

Web services builds on top of the development of the Internet. Figure 1 shows one view of the various connected standards around their development and deployment.

We have TCP/IP as the basic foundation of the Internet, which uses HTTP/HTTPS as ubiquitous communication protocols. With the definition of XML, a ubiquitous markup language has been created to describe and tag data. XML Schemas have recently surfaced as a rich alternative mechanism to DTDs, to define in XML data about data (metadata).

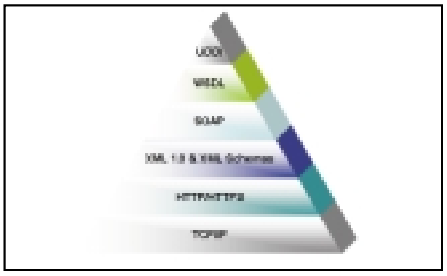


FIGURE 1 **Connected standards around Web services**

SOAP is the first entry into the world of Web services by defining a communication protocol using XML Messages for disparate systems to utilize the Web to communicate with each other. WSDL (Web Services Definition Language) provides metadata and a description of SOAP-based Web services; UDDI (Universal Directory, Description, and Integration) provides a standard registry and discovery mechanism to register, locate, and consume Web services.

SOAP

SOAP (Simple Object Access Protocol) is a standard which defines a collection of XML messages that can be used for intrasystem communications. It expands the World Wide Web, which has traditionally been used for information discovery and dissemination. SOAP allows the Web to be used as a common vehicle by systems to communicate with each other. These systems can be within a department or a corporate intranet, and communicated within corporate boundaries using the HTTP protocol, or can encompass B2B communications by providing the basic communication protocol for extended enterprise-based integrations.

Because the message format is based on the XML standard, SOAP can be used to communicate among multiple computer architectures, languages, and operating systems. SOAP enables a new class of applications called Web services, which expose services in a standard way so that application developers can create new applications by putting together services from many different sources on the Web.

SOAP defines an XML Schema–based Object Model for three main objects – SOAP Envelope, SOAP Header, and SOAP Body. Let's examine the various XML objects by looking at what happens behind the scenes when a client invokes the simple SOAP-based Web service, HelloService (see Listing 1).

WSDL

SOAP provides a standard way for systems to communicate with each other using the ubiquitous Internet protocols HTTP/HTTPS. No service consumer would be able to use Web services seamlessly unless they knew the exact specification or metadata about the SOAP service. WSDL, a specification developed by the UDDI group and also submitted to the W3C, provides such a way through an open XML-based syntax. WSDL allows service providers to expose the various Web services through an XML specification. Technically speaking, a WSDL document represents the description of a Web service as a collection of ports (or endpoints). The document is typically composed of two sections: one which describes the abstract data types of the messages and the operations, and the second a binding section which associates a service address (typically a URL) with the type definitions.

Typically, if you are a developer/architect creating Web services, then apart from providing the URI/URL of your SOAP service, you want to publish its specification (types/parameters) using WSDL. However, the Microsoft .NET toolset allows the WSDL-based service description to be almost prebuilt for you. Web services developed using ASP. NET (through .asmx files) provide an auto-generated and customizable WSDL structure.

UDDI

Where there is a description or metadata about a collection of objects, a natural extension to that philosophy would be to have a registry. UDDI provides both private and public registries of SOAP-based services defined using WSDL. UDDI is then, of course, a Yellow Pages to the world of dynamic Web services. The aim is to provide true B2B and Enterprise Integration. Potentially, a scenario like this can be conceived and developed where a computer manufacturing company wants to order some raw materials from B. It queries the public UDDI registry, hosted by a vertical exchange and through that is able to locate a company which provides the appropriate raw materials. Through well-defined services, an integration workflow is then able to call an order entry system of that supplier and order goods. This can be done in an automated, seamless fashion. If this scenario looks a little difficult to achieve, try simplifying some aspects of it and you would realize that the individual components themselves can prove to be very useful as well.

Types of UDDI Registries

Three major UDDI registry providers can be conceived:

1. ***Private Registry***: Your own organization may document and describe its internal Web services, which can foster inter-departmental communication and integration. Like any registry, you will also have separate Testing and Production registries as well. As we will discuss later, Microsoft provides a tool called UDDI Developer Edition 1.5 which can be used to create a private UDDI registry.
2. ***Public UDDI Registries***: As part of its commitment to promote the UDDI standard, Microsoft has created its own publicly available secure UDDI registries. The production UDDI Registry is available at http://uddi.microsoft.com; a test version of the registry is available at http://test.uddi. microsoft.com.
3. ***UDDI Registries hosted by Exchanges***: As part of your vertical or horizontal industry, a unification and common access initiative can potentially foster the development of a registry focused on your service offerings. Typically, these exchanges have something in common – either they are focusing on solving a particular business problem or are developed as part of a vertical organization.

Microsoft .NET Framework

Overview

From a developer's perspective, Microsoft .NET Framework is an environment for developing, deploying, and executing applications – both traditional Windows and Web-based applications as well as dynamic Web services. With a rich set of class libraries and the flexibility of multiple languages, .NET framework allows developers to focus on creating application business/presentation logic and leaving the "plumbing" work to the underlying framework.

.NET Framework was in Beta 2 (this stage is said to be feature-complete) at this writing. More information is available on the .NET homepage at www.microsoft.com/net/.

Components

The core Microsoft .NET Framework includes the common language runtime, class libraries and the bindings for the supporting languages (toolkits, document, etc.), and ASP.NET. In addition, the Visual Studio toolset (Visual Basic, Visual C++, etc.) has emerged under a common next-generation Visual Studio.NET umbrella. Under the .NET umbrella is a group of components, including Windows 2000 as the core operating system, .NET Enterprise Servers such as SQL Server 2000, and a set of Web services provided by Microsoft and collectively called the .NETMyServices (see Table 1).

TABLE 1

Component	Functionality
.NET Common Language Runtime	The core/common infrastructure
.NET Class Libraries	Set of class libraries for application developers to develop and deploy traditional Windows-based and Web applications and Web services
.NET Languages	C#, Visual Basic.NET, JScript.NET, Managed Extensions to C++ (for inter-operability with C++) and Visual J .NET, the newest binding from a language perspective
ASP.NET	Web Application Scripting Framework; also allows Web services development
ADO.NET	Database Access Services for .NET framework – set of classes that allows .NET applications to use enterprise databases
SOAP & Web Services	SOAP-based distributed services, alternative mechanism for RPC-styled remote object calls
COM+ (Component Services)	Provide a set of services for reliability and scalability to .NET applications – transaction processing, queued components, object pooling, role-based security, events, resource management, just-in-time activation, etc.
Visual Studio.NET	Development tool for creating .NET applications and components
Windows 2000 Servers	The core operating system; .NET framework will be released as an add-on for these platforms; Also, a new series of "NET Servers" are planned which will have the .NET framework as their foundation
.NET Enterprise Servers	Represents the server-side offerings available from Microsoft for .NET applications – Application Center 2000, BizTalk Server 2000, Commerce Server 2000, Exchange 2000, SQL Server 2000, Host Integration Server, Internet Security and Acceleration Server 2000, SharePoint Portal Server 2001, Mobile Information Server 2001, Content Management Server 2001

Tools for Developing Web Services Using .NET

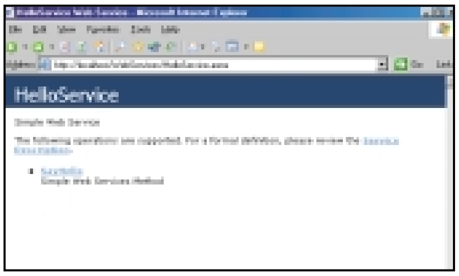
As part of the .NET umbrella, a number of tools will be available for development and deployment of Web services.

ASP.NET-based Web Services

A key feature of the next generation Active Server Pages (ASP.NET) technology is that it makes creation of Web services as simple as creating dynamic Web applications. ASP.NET-based Web services (written as .asmx files) aim to revolutionize the world of Web services. ASP.NET Web services intend to achieve the same benefits that Active Server Pages (and now ASP.NET) have achieved for creating dynamic Web pages. Developing and deploying a Web service using ASP.NET Web services support is as simple as developing and deploying a Web page.

Listing 2 creates a full-featured SOAP-based service called HelloService with a method SayHello that returns a string message. .NET Framework–based Web services use SOAP for communication and WSDL to represent the description and metadata for a service. An important benefit of this approach is that the various pre-built .NET libraries and the custom libraries that you develop can be used to create dynamic Web services. For instance, we can use the flexibility of ADO.NET-based components to interact with relational databases such as SQL Server. Through .NET and the COM interoperability layer it's possible to interact with any COM-based packaged enterprise application, such as SAP R/3 and Siebel, and make it available as an open Web service.

Simply opening the URL in Internet Explorer provides a basic rendering which lists the various methods and their descriptions (see


HelloService

Simple Web Service

[illegible]

- SayHello
  Simple Web Services Method

FIGURE 2 HelloService Web service

Figure 2). Executing the Web service with a browser using the URL http://localhost/WebServices/HelloService.asmx/ SayHello? returns the following XML response:

```
  <?xml version="1.0" ?>
  <string xmlns="http://www.silver-
line.com/Webservices">Hello World from
.NET </string>
```

Browsing to the URL http://localhost/WebServices/HelloService.asmx?WSDL, returns the description about the Web service in WSDL

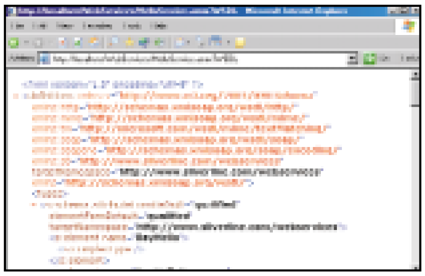


FIGURE 3 Description in WSDL format

format (see Figure 3).

The returned XML can then be processed by any application which can communicate using HTTP/HTTPS and has XML processing capability.

It's also possible to invoke the Web service as part of another .NET application after proxy classes have been generated for the appropriate language bindings. This mechanism is a .NET standards–based alternative for remote objects. As we will see below, Visual Studio.NET has built-in integration for discovering and using .NET Web services.

```
HelloService service = new
HelloService();
String msg = service.SayHello();
Console.WriteLine(msg);
```

Using the Windows XP Professional SOAP Client

The same service can also be invoked through any environment which supports SOAP/WSDL specifications. For instance. the upcoming Windows XP Professional ships with SOAP client software which provides easy access to SOAP/WSDL-based Web services. The following code snippet shows the basic script required to call the Web service:

```
Dim client
Set client =
CreateObject("MSSOAP.SOAPClient")
client.mssoapinit("http://localhost/WebS
ervices/HelloService.asmx?WSDL")
wscript.echo client.SayHello()
```

Visual Studio.NET

Microsoft Visual Studio has been the key de-facto standard for developing Visual Basic-, Visual C++-, and ASP-based applications. Visual Studio.NET (also known as Visual Studio 7.0) is the next revision of Microsoft Visual Studio toolset. Visual Studio.NET supports development in the new Microsoft language C#, Visual Basic.NET, and ASP.NET. Visual Studio.NET provides an integrated development and debugging platform for development of Windows forms–based GUI applications, Windows services, reusable components (or building blocks), Web applications, and services.

From a Web services perspective, Visual Studio.NET supports integrated development of C# and Visual Basic.NET–based ASP.NET development. Web services can be designed as easily as a Web application or even a traditional Windows form. Integration with supported databases (using ADO.NET) is key and enables faster wizard-based development of dynamic applications. These features are provided by Visual Studio.NET from Web services.

One key feature provided by Visual Studio.NET is that of adding a remote Web reference. This allows Visual Studio to be a service consumer and use the intended Web service within its own project. From an application perspective, it's possible for your business partner to establish a set of core SOAP/WSDL-based Web services and these services, including their complex parameters, can then be automatically called. This capability is provided by a wizard called "Add Web Reference" and is similar to adding a reference to a local DLL or .NET Component. The wizard can bind to either a remote/local WSDL service contract or a discovery page URL (.disco or .vsdisco files). The wizard also allows querying the Microsoft UDDI Registry (Test and Production) to find services for a given business name as well. For instance, our Web service, http://localhost/WebServices/

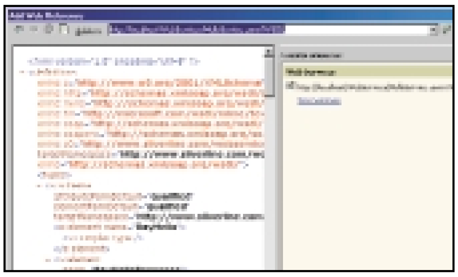


FIGURE 4 Remote Web reference

HelloService.asmx can be linked through its WSDL contract, http://localhost/WebServices/HelloService.asmx?WSDL, as shown in Figure 4.

The reference then shows up next to all of the component references and the Web service can be used within the project like any other class library. For instance, here's a code snippet using our previously defined Web service.

```
using System;
namespace DemoApp
{
 public class Main
 {
  public Main()
  {
      localhost.HelloService hello
                    = new
localhost.HelloService();

Console.WriteLine(hello.SayHello());
  }
 }
}
```

Microsoft UDDI SDK

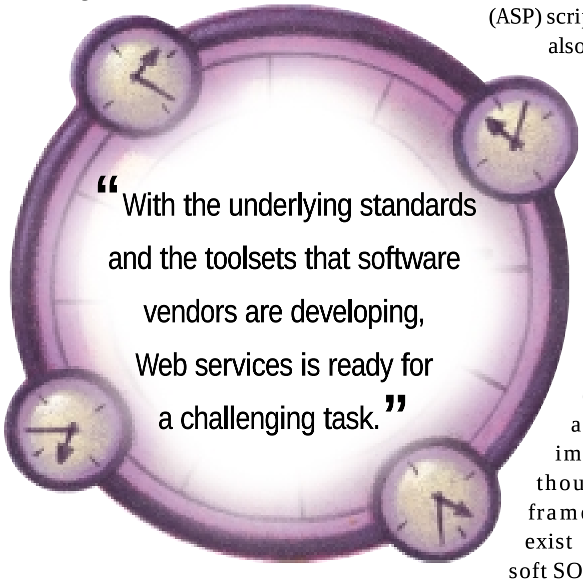
As part of Microsoft's public UDDI Registry implementation (both production and test registries) at http://uddi.microsoft. com, Microsoft has released a toolkit called Microsoft UDDI SDK which provides object mappings of the various UDDI functions. These objects can then be used as part of any software program/component; a basic UDDI Explorer application is available as part of the SDK. The current release (1.5.2) of the UDDI SDK contains two versions of the SDK, one to use from within the existing production-quality Visual Studio 6 toolset and another which targets the upcoming Visual Studio.NET Beta. It also contains a beta version of an implementation of a private UDDI registry, known as the UDDI Developer Edition 1.5. which requires a Windows 2000 operating system and Microsoft SQL Server (Personal, Standard or Enterprise; MSDE can be used as well).

Microsoft BizTalk Server 2000

One of the components of .NET Enterprise Servers, BizTalk Server is an integration platform for connecting internal enterprise systems with external customers, partners, and suppliers. BizTalk Server leverages XML in multiple fashions: XSLT for transformation of data as it flows from one system to another; XML-based Schemas to define the documents. It implements the XML/SOAP-based BizTalk Framework, which provides a core set of reliable messaging specifications for B2B communications. BizTalk Server supports multiple pluggable transport protocols, such as HTTP/HTTPS, SMTP, and Message Queuing, and may be used to extend Web services for asynchronous communications.

Microsoft SOAP Toolkit 2.0

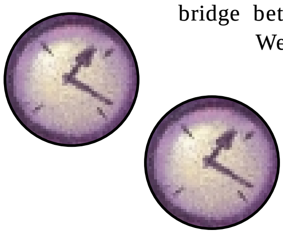
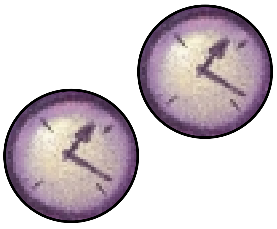
A number of technologies that we've explored here, particularly ASP.NET, are part of the .NET framework which is in beta release at this writing.


"With the underlying standards and the toolsets that software vendors are developing, Web services is ready for a challenging task."

However, you can start developing Web services using ASP/COM+ technologies. Microsoft has released Microsoft SOAP Toolkit, which will provide a bridge between the world of Web services (SOAP, WSDL, etc.) and the world of COM/DCOM components. Using a wizard-based approach, SOAP Toolkit allows Web services wrappers to be defined on top of existing COM-based components and then deployed using Internet Information Server via Active Server Pages (ASP) scripts or an ISAPI Listener. SOAP Toolkit also generates stub asp scripts which expose the WSDL interface of the Web services created.

Conclusion

With the underlying standards and the toolsets that software vendors are developing, Web services is ready for a challenging task. Microsoft .NET framework takes Web services forward by providing a number of integrated tools for building and deploying perspective. One important thing to note is that even though key components of the .NET framework are in beta stage, there exist key components such as Microsoft SOAP Toolkit and BizTalk Server that can help in Web services–enabling existing applications in preparation for .NET.

References

1.Web Services Description Language (WSDL) 1.1: www.w3.org/TR/wsdl
2. Simple Object Access Protocol (SOAP) 1.1: www.w3.org/ TR/SOAP/
3. Universal Description, Discovery and Integration (UDDI): www.uddi.org/specification.html
4. Microsoft UDDI SDK v1.5.2: www. microsoft.com/downloads/release.asp? ReleaseID=30880
5.UDDI Technical Overview http://uddi.microsoft.com/developer/techOverview.aspx
6. Microsoft .NET: www.microsoft.com/net
7.MSDN Online .NET Section: http://msdn.microsoft.com/net/
8.Microsoft Visual Studio.NET: http://msdn.microsoft.com/vstudio/nextgen/default.asp
9.MSDN Online Web Services Section: http://msdn.microsoft.com/webservices
10.Microsoft SOAP Toolkit 2.0 SP2: http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/msdn-files/027/001/580/msdncompositedoc.xml
11.Microsoft BizTalk Server: www.microsoft.com/biztalk/
12.Microsoft BizTalk Framework 2.0: www.microsoft.com/biztalk/techinfo/framwork 20.asp

Listing 1

```
Request XML
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SayHello
xmlns="http://www.silverline.com/Webservices"/>
  </soap:Body>
</soap:Envelope>
Response XML
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SayHelloResponse
xmlns="http://www.silverline.com/Webservices">
      <SayHelloResult>Hello World from
.NET</SayHelloResult>
    </SayHelloResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 2

```
<%@ WebService Language="C#"
class="Silverline.HelloService" %>
namespace Silverline
{
 using System;
 using System.Web.Services;

[WebService(Namespace="http://www.silverline.com/Webservice
s",
        Description="Simple Web Service")]
 public class HelloService : WebService
 {
  [WebMethod(Description="Simple Web Services Method")]
  public String SayHello()
  {
   return "Hello World from .NET";
  }
 }
}
```

Download the code at
sys-con.com/webservices

2002

JUNE 24-27

JACOB JAVITS CONVENTION CENTER

NEW YORK, NY

JAVA, XML AND .NET TECHNOLOGIES

web services EDGE conference & expo™

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

JDJ EDGE conference & expo™

INTERNATIONAL JAVA DEVELOPER CONFERENCE & EXPO

XML EDGE conference & expo™

INTERNATIONAL XML CONFERENCE & EXPO

Fundamentally Improving the Speed,
Cost & Flexibility of Business Applications

Who Should Exhibit...

Java, XML, Web services and .NET Technology vendors, staking their claim to this fast-evolving marketplace.

JDJEdge 2002 and Web Services Edge East 2002 Will Feature...

- Unmatched Keynotes and Faculty
- Over 150 Intensive Sessions and Fast Tracks
- The Largest Independent Web Services, Java and XML Expos
- An Unparalleled Opportunity to Network with over 5,000 *i*-technology professionals

Who Should Attend...

- Developers, Programmers, Engineers
- i-Technology Professionals
- Senior Business Management
- Senior IT/IS Management
- Analysts, Consultants

THE CALL FOR PAPERS IS NOW OPEN
VISIT WWW.SYS-CON.COM FOR DETAILS

James Gosling,
The Father of Java

FOR MORE INFORMATION

SYS-CON EVENTS, INC.
135 CHESTNUT RIDGE ROAD
MONTVALE, NJ 07645

201-802-3069
WWW.SYS-CON.COM

SYS-CON MEDIA
OWNED & PRODUCED BY
SYS-CON EVENTS

Web Services and ZOOBA

A software model opens the door to new solutions

In the history of technology, the growth rate of the World Wide Web (W3) was unprecedented. The Internet spread faster and matured more rapidly than the TV, the radio, or even the light bulb. It was the "big bang" phenomenon of information. W3 helped transform global business into a giant, borderless exchange, with information valued as currency.

In the course of its rapid development, the principles of distributed computing, component engineering, and reuse were very important. That's evidenced by proposals ranging from application-to-application integration (A2A), business-to-business commerce (B2B), and collaborative computing to message-oriented-middleware (MOM), *n*-tier architecture, object-oriented analysis and design (OOA and OOD) techniques, and more. Some of those initiatives have been successfully implemented, including the component object model (COM) and Java 2 Enterprise Edition (J2EE), Enterprise Java Beans (EJB), Java Message Services (JMS), and Java Naming and Directory Interface (JNDI), as well as CORBA Component Model (CCM) and others. All offer very useful answers for solving specific problems.

The need to address certain high-level issues makes it necessary to develop a more encompassing framework and requires us to investigate the field of distributed computing on a macro-level. Moreover, there are many assumptions used in current models. By questioning those assumptions and looking at the problems from a different angle, we're encouraged to seek a new model.

In this article, I'll discuss Web services and P2P technologies and try to answer the following questions:

1. What are Web services?
2. Why are they steering us in a new direction?
3. Where do they connect with standards like Java and XML?
4. Which established conventions will they affect and likely transform?
5. How are they creating new paradigms for distributed computing?

We'll also look at a new software integration model called ZOOBA, which stands for eXtended Object-Oriented Broker Architecture (see Figure 1).

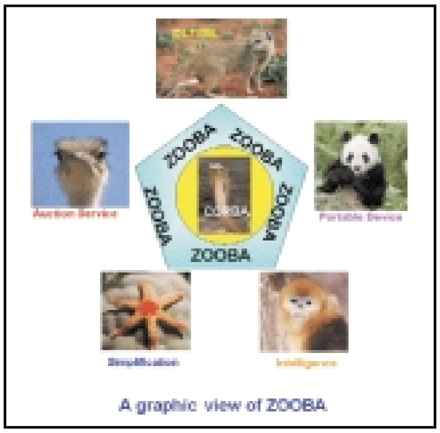


FIGURE 1 **Visualizing ZOOBA**

What Are Web Services and P2P?

Web services embodies a new approach to reusing, component engineering, and distributed computing issues from both technical and business perspectives. It's based on the notion that software, such as systems, applications, programs, and other computing activities concerning objects, messages, and documents, can be modeled into services. Large events can be broken down into smaller discrete activities, while small actions can be grouped to form units of work. In general, any software can be transformed into Web services if it exhibits the following characteristics:

1. It provides interface(s) through which other programs can invoke it.
2. It is registered with and can be located by one or more service registries.

It's also important that Web services rely on standard technologies. For example, XML is recognized as the lingua franca for information on the Internet, and HTTP is the default protocol for sending and receiving information for many applications and browsers. Web services depend on the following technologies:

- Simple Object Access Protocol (SOAP; see Listing 1)
- Web Services Description Language (WS DL)
- Universal Description, Discovery, and Integration language (UDDI)

Electronic Business XML (ebXML) is similar to SOAP/WSDL/UDDI, and is also an important ingredient for Web services. Web services' default transmission protocol is HTTP [though FTP and e-mail protocols are possible alternatives].

The current concept of distributed computing is based on a server-centric model. It originated in the IBM mainframe model, where numerous dumb terminals were attached to a powerful mainframe. It grew into a client/server model, multiple workstations connected to server(s). The current W3 model is a browser-based version of the server-centric model, where dumb terminals have been replaced by Web browsers and mainframes by Web servers. The server-centric model's strength lies in the centralization of data, but it has drawbacks. Those issues become particularly clear when eight abilities are considered: scalability, availability, manageability, compatibility, security, stability, portability, and flexibility.

The peer-to-peer (P2 P) model approaches distributed computing froma completely different angle. Under P2P,

Author Bio

Raymond Gao is an e-commerce architect with Sun Microsystems, Inc.'s dot-com practice. His primary areas of work involve Java, XML, Linux, and object-oriented technologies for e-commerce architecture and distributed computing projects at the enterprise level. Ray has spoken at technology conferences and has contributed articles to leading journals as well as authored numerous white papers. He also wrote the popular CoffeeCup utility.

RAYMOND.GAO@CENTRAL.SUN.COM

every computing element is treated as parallel and there's no predefined hierarchy. A computer can be both the server and the client. A simple example is a music- or file-sharing community. When a user downloads a file, it plays a client role. When that user redistributes the file, it changes to a server role. Project JXTA (Juxtapose) is an example of the P2P model.

The Paradigm Shift

Web services and P2P models are disruptive technologies that represent a paradigm shift for distributed computing. While Web services help reduce chaos by providing a standardized directory service as well as coordinating transactions and interactions between end points, P2P is an improvisation-based model. That's a radical departure from the deterministic model on which CORBA is based. Under P2P, there are no predefined end points for Web services.

Let's look at a hypothetical Customer Relationship Management (CRM) application. Mr. Johnny Needata needs to respond to his client's request for quotes. He wants to use his desktop CRM tool to retrieve the latest quote. Ms. Jane Hathdata is his peer on the company's local network who just downloaded the latest sales quote to her workstation from a remote headquarter server for a different deal. If the P2P capability is enabled on both their machines and the central server is busy or under heavy load, Johnny's CRM application can quickly fetch information from Jane's computer located on the same local network and avoid the server bottleneck problem. As the number of users increases, there'll be more copies of data on the network, and the available data sources will multiply. In other words, the increased load may enhance the overall performance of the complete system. This is in sharp contrast to the server-centric model, where the system's performance degrades as the load increases.

This doesn't invalidate CORBA. It only shows that the model needs to be extended so it will function for nondeterministic situations or it will risk becoming obsolete. ZOOBA is a tower that builds on the CORBA foundation, retaining all existing features and supporting several new services, including the following:

1. Auction service
2. Intelligent direction service
3. Simplification service
4. DL2XML
5. Support for portable and networked device aware (PANDA) service
6. Collection service
7. Concurrency service
8. Event service
9. Externalization service
10. Interoperable naming service
11. Licensing service
12. Life cycle service
13. Notification service
14. Persistent object service
15. Property service
16. Query service
17. Relationship service
18. Security service
19. Time service
20. Trading object service
21. Transaction service

Re-examining the CRM example, ZOOBA's advantage is apparent. Instead of binding to a single, specific host, the CRM tool's Object Request Broker (ORB) can advertise a request for data on the network, using a protocol such as IP broadcasting. Based on feedback messages, the ZOOBA model can facilitate intelligent decision-making for information retrieval based on any number of criteria: resource cost, network speed, security settings, data reliability, locality, geography, time, and length for the operation.

Web Services' Key Components

You must have a good understanding of Web services to succeed in building reusable service components. That knowledge will help you classify modules based on functions and construct a component model. Elements of Web services can be grouped into three classes with respect to the roles they play:

1. Web services metarepository
2. Web services engine
3. Web services platform

The Web services metarepository is the registry for publishing and advertising the availability of Web services. UDDI is a prominent implementation. It behaves like a phone book with categorized listings of companies or entities:

- The ***White Pages*** include information about an entity's address (possibly in URL format), contacts, and known identifiers.
- The ***Yellow Pages*** list companies and their services under standard industry categorizations.
- The ***Green Pages*** keep the technical information about Web services' entry points and URLs to various information discovery mechanisms. Such information includes, for example, unique IDs, file locations, descriptions about the service, and links.

The metarepository holds metadatas, pointers to actual services implemented by the Web-service engine. It's analogous to the factory pattern used in OOA and OOD. It churns out "objects" (URLs, pointers, and handles to services' APIs) without needing users to specify which "classes" (types of services) to use. While the Web service metarepository is the facade, the Web service engine is the service implementation. It holds the actual service logic and performs in accordance with requests.

Web services translates into distributed computing. Because it's a good practice for distributed systems to be loosely coupled, it's important to not expose Web services' internal properties and operations to the outside unnecessarily. This model is similar to EJB, which has the home interface (metadata), the remote interface (handles to APIs), and the bean class (implementation). Web service metarepositories can also offer additional functions, such as policy and rule management, context awareness, and workflow automation features. The J2EE model calls for a container that manages EJB's persistence, life cycle, transaction properties, and other attributes. The Web service platform provides infrastructure-level functions for Web services modules. It needs to address common-denominator issues, such as binary compatibility, interoperability, portability, state and session management, message delivery, and dynamic scaling. Dynamic scaling is more practical than the conventional concept of scaling because:

1. The demand for Web services tends to be unpredictable.
2. There are costs associated with adding capacity and resources.
3. Business is driven to optimize the use for all resources.

Web services provides a model that adjusts to its surroundings and responds to changes in external conditions – i.e., using high performance resources during high demand and shifting to low-cost resources when demand is low.

New Directions

There are some good Web sites dedicated to Web services and related technologies:

- www.uddi.org
- www.w3.org/TR/wsdl
- www.sun.com/sunone

- www.ebxml.org
- www.oasis-open.org
- www.w3.org/TR/SOAP
- www.unece.org/cefact
- www.bpmi.org
- www.xmethods.net

By examining what has occurred in the market for Java, XML, HTML tools, and other areas of the Internet, you can see the likely future of Web services. UDDI and WSDL create an interface-driven model so multiple Web services providers can provide seemingly identical Web services using different implementations. Web services could become commodities, leading to the creation of a Web services marketplace sponsored by service-registrars. "Web services commoditization" means that the only way to differentiate one provider's product from another's is cost, although providers will try to distinguish their offerings by stressing quality, reliability, innovation, and other advantages.

There's been a lot of talk about the convergence of Java and XML technologies. That trend can be observed in the increasing effort to develop persistence mechanisms that bridge the gap between the "noun" (XML) and the "verb" (Java) of Internet computing, as shown by Java APIs for XML – Java API for XML Processing (JAXP), Java API for XML Data Binding (JAXB), Java API for XML Registries (JAXR), Java API for XML Messaging (JAXM), and Java API for XML-based RPC (JAX-RPC) – along with XML Meta Interchange (XMI). Since J2EE is one of the primary models supported by the OMG, those efforts, in conjunction with IDL2XML, could expand the operational envelope for distributed computing.

Conclusion

P2P, Web services, and ZOOBA will become significant technologies in the next wave of Internet development. Although still in their early stages, they show that distributed computing doesn't need to be hierarchical and static. Instead, a dynamic, fluid, and "democratic" model will be a viable alternative. This new framework could use resources more efficiently because it stresses improvisation; and our example shows it can scale intelligently according to demand. Changes on the Internet can happen quickly – and unexpectedly. The self-regulated system will excel because it tends to be the most flexible and adjustable solution.

Suggested Reading

1. "Open Net Environment (ONE) Software Architecture, An Open Architecture for Interoperable, Smart Web Services." Sun Microsystems, Inc., 2001.
2. "Web Services White Paper: A Technical Overview." IONA Technologies, PLC, March, 2001.
3. "XML Metadata Interchange (XMI)," Sponsored by OMG.
4. Bartlett, D. "CORBA Component Model (CCM):, Introducing Next-Generation CORBA." IBM Developer Works, April, 2001.
5. Gao, R. "The Next-Generation Internet." *eAI Journal*, May 2001.
6. Gao, R. "JMS, A Portable Framework for Connecting the Enterprise." *eAI Journal*, November/December, 2000.
7. Cobb, E.. "CORBA Components: The Industry's First Multi-Language Component Standard." BEA Systems, presentation at OMG meeting Oslo, Norway, June 16, 2001.
8. "XML Protocol Comparisons." A Web document hosted by W3.org's XML Protocol Working Group

Listing 1 Example SOAP Request and Reply

```
Request
XML Prolog <?xml version="1.0" encoding="UTF-8" ?>
SOAP Envelope
(Open) <SOAP-ENV:Envelope
 xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingSytle="http://schemas.xmlsoap.org/soap/encoding/"
>
Header (optional)<SOAP-ENV:Header>
<tns:Transaction
 xmlns:tns="urn:CanPerformWebServices"
 SOAP-ENV:mustUnderstand="1"
>
</tns:Transaction>
</SOAP-ENV:Header>
Body <SOAP-ENV:Body>
<schd:SetupAppointment
 xmlns:schd="urn:OnlinePetWebServices"
 SOAP-
ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
 <PetName>ZOOBA</PetName>
 <Date>April 8th, 2002</Date>
 <Time>10 A.M. </Time>
 <TypeOfService>vaccination</TypeOfService>
</schd:SetupAppointment>
</SOAP-ENV:Body>
SOAP Envelope
(Close) </SOAP-ENV:Envelope>

Reply
XML Prolog <?xml version="1.0" encoding="UTF-8" ?>
SOAP Envelope
(Open) <SOAP-ENV:Envelope
 xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-
ENV:encodingSytle="http://schemas.xmlsoap.org/soap/encoding/"
>
Header (optional)<SOAP-ENV:Header>
<tns:Transaction
 xmlns:tns="urn:CanPerformWebServices"
 SOAP-ENV:mustUnderstand="1"
>
</tns:Transaction>
</SOAP-ENV:Header>
Body <SOAP-ENV:Body>
<schd:SetupAppointmentResponse
 xmlns:schd="urn:OnlinePetWebServices"
 SOAP-
ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
 <Result>Confirmed</Result>
 <DetailResult>
  Vaccination for ZOOBA at April 8th, 2002, 10 A.M.
 </DetailResult>
</schd:SetupAppointmentResponse>
</SOAP-ENV:Body>
SOAP Envelope
(Close) </SOAP-ENV:Envelope>
```

Download the code at
sys-con.com/webservices

COMMENTARY

Taking a key role in offering business ROI

Using **ebXML** & **Web** Services

WRITTEN BY

David RR Webber

David RR Webber is co-author of EbXML, the New Global Standard for Business on the Internet (New Riders Press) and VP, business development for XML Global Technologies, Inc. (www.xmlglobal.com) He is responsible for the expansion and implementation of XML Global's strategic business development initiatives. He has more than 20 years of experience in designing and implementing XML- and EDI-based electronic business systems for a wide range of industries and has done extensive work for various government organizations on XML and e-business standards.

The fundamental notions behind Web service implementations are not new. In fact, many people would argue that their existing Internet RPC systems are Web services. What is different is the use of XML, and XML messaging structures based on SOAP, to control and support these services. This provides a consistent, open means of interaction that is easy to understand and assimilate; and this is what drove the original HTML content revolution and the birth of the Web itself.

Consistent business interactions are the very essence of what the ebXML specifications are all about (www.ebxml.org) and provide the means for simple and ubiquitous global commerce through the use of affordable XML-based infrastructure. Given the level of commitment from significant industry groups and government sponsors, the adoption of ebXML will reach critical mass during 2002. A key factor in this is organizations needing to adopt open, mature, and stable XML technology that will position them not just for current requirements but for long-term future needs. Providing the basis of this are the ebXML messaging services known as TRP (Transport, Routing, and Packing) that ensure consistent and dependable exchange of content and services.

When these ebXML message handling service (MHS) specifications were developed, keen eyes immediately saw a broader relevance for the enveloping and SOAP-based interchanges than just replacing old-style EDI package services. Indeed, Web service–based implementations of ebXML MHS can already be seen at sites such as www.catXML.org, where catalogue information consolidation and remote querying are available using ebXML-compliant SOAP messaging systems as the basis for a dynamic query/response system.

As people implement Web service interfaces and seek to contribute them to the overall body of the Web, they realize that they need to define details about those services. If the service is e-business related, then the natural place for this is ebXML. The foundation of the ebXML process itself is fundamentally built around another key piece of the puzzle: registry services. When you consider the needs of messaging services, you quickly see the key role that a registry provides to a community of interest within industry or trading partner networks. The registry serves up consistent and reliable exchange information on definitions, content, process steps, and participants' profiles that allow smooth and efficient business processes to be built and deployed. These are vital facets of long-term, sustained, and dependable Web services.

While XML provides the means to have consistent exchanges, this is not enough by itself. The semantics and meaning of the actual content, and how and where the Web service interface functions are, are required prerequisites to consistent interoperability.

The ebXML specifications provide all of the artifacts needed to enable e-business systems within the context of Web services. In fact. an excellent example is the bootstrap for registry services themselves. The ebXML registry services also use the ebXML MHS system to provide direct machine services. Within this context, you can define the support the registry provides, its own profile of standard queries and responses it implements, and then the means to bootstrap the registry so that machine-addressed interfacing to the registry can be configured. A typical implementation is showcased at http://registry.xmlglobal.com; this software represents the first commercially available ebXML registry system. Other ebXML registries are available as code bases from Sun and IBM, and at least two more software vendor companies are working on their own commercial product releases as well.

The other major player in the arena of directories of Web services is the UDDI initiative. The UDDI approach to this integration puzzle is the use of tModel definitions. Work is underway to make tModel and ebXML specification definitions for common e-business artifacts interoperable. The goal is to align the ebXML and UDDI work on directories. UDDI adds significant value above an ebXML-based system by providing the business directory of Yellow and Green pages definitions. The ebXML system itself provides the robust e-business interchange mechanisms and open standards across industries and platforms that are required for the physical XML-based application-to-application (A2A) and business-to-business (B2B) low-level couplings. This includes such things as business process and information transformation scripting. Vendors are now scrambling to differentiate their existing EAI (Enterprise Application Integration) toolsets and promote enhanced XML and ebXML processing capabilities to augment their existing suite of tools. One of the first to announce this was iWay Software (www.iwaysoftware. com), an Information Builders company, who released their XML Transformation Server (XTS) as part of their Enterprise Integration Suite in November 2001. XTS embeds XML Global's (www.xmlglobal.com) data transformation engine, which includes Web services and registry interfaces.

The whole area of supporting Web services and providing rich and consistent interoperability specifications for these is only just beginning. Key players such as ebXML can immediately provide value in this arena, and work is already underway to enhance the registry specifications to provide augmented tools specifically for Web services. More information on this work is available from the registry development team Web site at www.oasis-open.org/commi ttees/regrep.

Larger businesses and government departments can be expected to take advantage of the opportunities that consistent specifications provide to roll out enabling business interfaces for their larger communities of customers. Providing reliable, controllable, and consistent interfaces will be key factors for them. In this context, ebXML will clearly play a key role in allowing businesses to offer return on the their investments in XML-based technology infrastructure.

Written by Mike Richardson

An Introduction to WSDL

A critical part of an automated Web

The Web Services Description Language (WSDL) is an industry-standard XML vocabulary used to describe and expose services for client consumption. To be more precise, it's a simple, extensible, and reusable interface definition language that serves to specify Web service operations, messages and their data types, and the deployment details of network services. This article introduces the basic semantics of WSDL and illustrates its importance as a key element among the collection of technologies that have made Web services a reality. It requires a working knowledge of XML and XML namespaces. In addition, some familiarity with XML Schemas and SOAP is helpful.

Web services represent an important new breed of distributed Web applications. They are self-contained, self-describing, modular applications that can be published, located dynamically, and invoked across the Web. Utilizing distributed Web services, applications can be engineered, assembled, and deployed across Web application boundaries, regardless of the underlying hardware or operating system.

A Move to a Semantic Web

For a number of years, the majority of distributed systems have been designed using the most popular distributed component object models, such as CORBA, COM+, and Enterprise JavaBeans. Although these models have contributed to the success and growth of the Web, they have several unfortunate limitations. For example, network services exposed by COM+ interfaces utilize an RPC-based network representation, whereas Enterprise JavaBeans are commonly invoked over the RMI/IIOP protocol. This ultimately leads to a division among component models, and the inability to interoperate at the most fundamental level. The essential piece of the puzzle that's been missing for a number of years has been the motivation to design a standard set of APIs for intercomponent discovery, description, and communication.

Most experts are predicting a second wave in the growth of the World Wide Web and a shift towards a more automated and collaborative environment. Web services will be a major driving force behind this shift, the transformation into a "semantic" Web composed of self-describing resources and automation interfaces. However, for this trend to become reality, a few fundamental questions must be answered. For example, how can business and process logic, which may currently reside deep in internal corporate structures, be exposed on the Web in a standardized manner? In technical terms, we need to agree on a common set of semantics for describing Web service "interfaces." This is the motivation driving the development of WSDL.

As you continue to experiment with this new XML-based technology, keep in mind that the major proponents of WSDL and Web services in general have attempted to leverage as many existing standards as possible. For example, many SOAP implementations are simply a combination of XML and HTTP. In the same fashion, WSDL is a simple, human-readable markup language. If you have experience manipulating XML documents, then the learning curve should be minimal.

Although there have been numerous efforts to standardize on a service description language for a component-based Web, such as WebMethods' Web Interface Definition Language (WIDL) and Microsoft's Service Contract Language (SCL), the WSDL standard has generated the most support from technology vendors, including Microsoft, IBM, Ariba, and Sun. For example, the majority of Web services toolkits and component application servers have or will soon have built-in support for creating and manipulating WSDL documents.

Interface-Based Programming

From an application development standpoint, the ability to program against a set of abstract interfaces has been central to the development of most distributed object models. COM+ requires the compilation of IDL into a binary format to allow for intercomponent communication, and EJB relies on the generation of home and component interfaces. In addition, interface-based programming can serve as the foundation for component versioning, run-time–type inspection, location transparency, and polymorphism. WSDL expands on this tradition by providing an extensible description of component interaction that is both reusable and extremely flexible. Presenting software services through an interface-based WSDL vocabulary has several important ramifications in addition to presenting components in a standard fashion.

First, it should be noted that WSDL promotes the physical separation of a Web service's interface from its corresponding code implementation. In other words, the entry point to a component will be described in a WSDL document, but the actual code base that represents the business logic can be composed in any software language. The benefits of this should be evident. By keeping these two entities physically separated, a developer can continue to enhance the language specific implementation without changes to the external interface. In

Author Bio
Mike Richardson is a cofounder and the CTO of Distributed Objex, Inc. (www.distributedobjex.com), an IT solutions, training, and mentoring firm. His firm designs and implements major information systems for numerous companies in the Fortune 100. They have implemented early Web Services technologies at numerous client sites, such as SOAP, WSDL, and "pluggable providers".
MRICHARDSON@DOXINC.COM

addition, a WSDL document is extensible enough to support various network and messaging protocols, such as HTTP, SMTP, and FTP. This type of programming allows for ease of distribution, reusability, and enhanced versioning capabilities. An easy way to remember this type of model is to think of an interface or WSDL document as a contract language. The server states that it promises to implement the messages described in the WSDL document. In return, a client will consume and transform the WSDL document metadata into a language that will ultimately be understood and processed regardless of the underlying software and hardware implementation.

Second, when illustrating this type of separation between interface and implementation, Web services can be categorized as part of a "service-oriented" design paradigm. The WSDL document defines a component or procedure in terms of a service that the client wishes to invoke. Service-based architectures are different in many ways from traditional object-oriented models and are implemented in popular connectionless protocols, such as JMS (Java Messaging Service), HTTP, and asynchronous programming techniques.

The Need for New Approaches

Your development team may need to alter the way they approach problem domains, focusing on more granular and stateless programming models as opposed to components, which possess a reliance on state information. For example, a stateless session EJB used as a Web facade object would lend itself nicely to this type of programming paradigm.

Service-based models are inherently transactional, and scalability requires serious forethought rather than afterthought. When services respond in a highly procedure-oriented manner, they are often categorized as *coarse-grained* components. Coarse-grained components are a fundamental precursor to system scalability. As a technologist, you'll see that your information systems in the future will have a much higher reliance on technologies, such as messaging systems and workflow-centric technologies. For example, IBM's Web Services Flow Language and Microsoft's XLANG support workflow processes and programming models.

WSDL Structure

Now that we have examined some of the background and benefits of WSDL, let's move forward and inspect the structure and semantics that serve as its foundation. First, I'll present a general description of some of the various entities that comprise a simple WSDL document, followed by a more advanced look at each of the structures. Although most tools and service containers will have built-in support for automating the manipulation of WSDL, there are certain circumstances that will require a more intimate understanding of its syntax. For example, the marshalling of objects between Web services will most certainly require a deeper knowledge of WSDL.

As I previously stated, WSDL is highly extensible. For example, it doesn't require a certain network protocol or messaging format. However, for the purpose of this article we'll assume that we're using WSDL in a system whose message data conforms to the XSD Schema specification and the SOAP protocol.

From one perspective, there are actually two logical parts that combine to form a WSDL document – a reusable part and a non-reusable part. Most distributed component object models require the use of some type of registry or location service, such as an ORB. Web services don't require such interposition protocols, but the absolute location of a service endpoint must be declared in the non-reusable section of a WSDL document known as a *port*, which will be described in more detail later. The reusable piece of a WSDL document will allow applications to be deployed in multiple locations without duplicating interface definitions and remain extensible with regards to messaging formats and protocol bindings.

At first glance, the different elements that make up this structure may seem confusing and make manual interaction appear difficult and error-prone, which explains the influx of WSDL APIs such as JWSDL and programming tools such as Microsoft's SOAP Toolkit. However, it's helpful to envision a WSDL document as an abstract representation of a messaging system, with each of the seven parts serving as a component or subsystem. As we examine the various parts of a WSDL document, I'll make frequent analogies to a typical electronic mail system. Like WSDL, most mail systems have certain types of acceptable data (text, HTML, RTF). This data is further combined to form messages, which are sent and received by operations. Operations are then carried out using specific protocols, such as SMTP. Similar to other network protocols, ports and binding mechanisms must also be employed.

In order to construct a WSDL document, we first need to describe the basic data types that will be used in transmitting our messages. Most applications will require data types to conform to the XML Schema standard. However, WSDL supports the description of complex data types. The document fragment in Listing 1 illustrates how schemas can be utilized within WSDL to define types.

The root element of all WSDL documents is represented by the <definitions> element. In Listing 1, we've declared the WSDL namespace as the default namespace, so all of the elements are included in this namespace, unless otherwise specified with an additional declaration. Because XML Schema data types are defined by an abstract model, there are several rules that should be implemented when using XML Schemas; these include using elements instead of attributes, protocol-agnostic data types, and the use of xsd: anyType if the type is not important.

WSDL Operations

After the types used in an application are described, they are combined to form messages. WSDL messages don't rely on any specific network protocol. In other words, messages can ultimately be sent using SOAP, HTTP GET, SMTP, etc. Furthermore, in the typical Web-based system (request/response) messages will have to be described for input as well as output operations. Input and output operations will most likely need to accept parameters. These parameters are described within the <message> element and are known as message parts. The following document fragment illustrates message structures and parts in WSDL:

```
<message name="ClaimRequest">
 <part name="claimID"
type="xsd:string"/>
</message>
<message name="ClaimResponse">
 <part name="StatusCode"
type="xsd:string"/>
</message>
```

Each of the above message elements represents a function in a request/response pair described by the WSDL document. This collection of messages is known as an operation. There are only four forms of operations with built-in support in WSDL: one-way, request/response, solicit/response, and notification. In a typical mail system, sending a piece of mail would be classified as an operation. Transactional in nature, operations represent an all-or-nothing, single unit of work for the service being described. This is evidence that Web services truly represent a unique programming paradigm. Unfortunately, many experienced developers will be tempted to use WSDL as a simple object serialization format. This isn't the intent of a service-based Web infrastructure.

The cornerstone of a scalable, service-based programming model will be composed of messaging algorithms and stateless, granular procedural invocations rather than complex object models. When e-mail is sent to a co-worker, it either arrives or is never sent; there's no in-between. As I noted earlier, Web services will most likely cause an increased reliance on messaging, workflow-based tools, and languages. Microsoft's BizTalk Server and the Java Message Service (JMS) are examples of this trend. Sample code expressing operations in WSDL is shown below:

```
<operation name="GetClaim">
 <input message="ClaimRequest"/>
 <output message="ClaimResponse/">
</operation>
```

Ports and Port Types

WSDL documents can contain any number of operations. This collection is referred to as a *port type*. Port types fundamentally represent the set of functions that are publicly exposed by the Web service. They are defined in WSDL using the <portType> element:

```
<portType name="GetClaimService">
 <operation name="GetClaim">
  <input message="ClaimRequest"/>
  <output message="ClaimResponse/">
 </operation>
</portType>
```

Although messages can be represented by abstract data types, eventually they must be implemented in a concrete fashion. In other words, there still needs to be a physical wire format for representing your data in the physical world. Bindings are used to describe these concrete implementations, as seen in Listing 2.

The last and final data structure in a WSDL document is known as a port. Ports are non-reusable and are used to associate a binding with a protocol-specific address. In addition, they describe what are commonly referred to as SOAP endpoints. The following markup describes a port:

```
<port name="InsuranceClaimsPort" bind-
ing="GetClaimServiceBinding">
 <soap:address
location="http://www.doxinc.com:8080/so
ap/servlet/rpcrouter"/>
</port>
```

Although the preceding example implements the HTTP protocol, keep in mind that WSDL avoids limiting the number of network protocols. The WSDL specification describes a standard binding mechanism used to attach a specific protocol, data format, or structure to an abstract implementation, operation, or port. WSDL supports the creation of custom bindings and is also the preferred representation of a UDDI (Universal Description, Discovery, and Integration) Service Description element.

Conclusion

WSDL is a critical part of an automated Web based on services. In the spirit of SOAP and UDDI, WSDL is a simple language and it does a good job at leveraging existing technologies such as XML, XSD, and RDF (Resource Description Format). The use of WSDL promotes service-based design and process patterns, reusability, and dynamic component discovery. The current official state of WSDL can be found at www.w3. org/tr/wsdl.

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ClaimServer-interface"
  targetNamespace="http://www.doxinc.com/ClaimServer-
         interface"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
 <xsd:schema
targetNamespace="http://www.doxinc.com/claim.xsd">
  <element name="…">
  </element>
 </xsd:schema>
</types>
</definitions>
```

Listing 2

```
<binding name="GetClaimServiceBinding"
type="GetClaimService">
 <soap:binding style="rpc" transport="http://schemas.xml-
soap.org/soap/http"/>    <operation name="GetClaim">
  <soap:operation soapAction="http://www.getclaim.com/
         getclaim"/>
  <input>
   <soap:body use="encoded"
    namespace="urn:claim-requests"
    encodingStyle="http://schemas.xmlsoap.org/soap/
         encoding/"/>
  </input>
  <output>
   <soap:body use="encoded"
    namespace="urn:claim-requests"
    encodingStyle="http://schemas.xmlsoap.org/soap/
         encoding/"/>
  </output>
 </operation>
</binding>
```

Download the code at
sys-con.com/webservices

SHOP ONLINE AT JDJSTORE.COM FOR BEST PRICES OR CALL YOUR ORDER IN AT 1-888-303-JAVA

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

GUARANTEED BEST PRICES
FOR ALL YOUR
WEB SERVICES
SOFTWARE NEEDS

JDJStore.COM

CAPE CLEAR
$445.00 CapeStudio

CapeStudio is a Rapid Application Development (RAD) environment for creating Web Services and XML-based applications. It consists of an integrated toolset. CapeStudio Mapper enables XML data transformations in a graphical environment and generates standard XSLT. CapeStudio WSDL Assistant generates Web Services client proxy and server skeleton code based on standard WSDL. UDDI Registry Browser searches for available Web services descriptions.

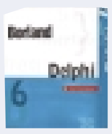
ALTOWEB
$3,540.00 Application Platform Release 2.5

The AltoWeb Application Platform lets you build, deploy, and manage J2EE applications and Web Services up to 10x faster without requiring extensive J2EE or Web Services expertise. How? By replacing lengthy, custom and complex J2EE, XML and Web Services coding with rapid component assembly and reuse.

CAPE CLEAR
$950.00 CapeConnect Three

CapeConnect is a complete Web Services platform that allows you to automatically expose your existing Java, EJB and CORBA components as Web Services without writing code. CapeConnect is capable of connecting a wide range of technologies (including Java, J2EE, CORBA, C++, COM, C#, Visual Basic, and Perl) using Web Services standards such as SOAP, WSDL, and UDDI. CapeConnect can be used internally over intranets and also can be used to expose business logic over the Internet, for use by customers and partners.

BORLAND
221980 $2,999.00 Sale Price $2,849.99 Delphi



Delphi 6 makes next-generation e-business development with Web Services a snap. BizSnap Web Services development platform simplifies business-to-business integration by easily creating Web Services. DataSnap Web Services-enabled middleware data access solutions integrate with any business application. Check out our Web site for system requirements.

SITRAKA
$3,999.00 Sale Price $3,799.00 JClass ServerChart V1.1

If you've used JClass Chart in your client-side applications, you already know the value of data visualization to your end-users. With JClass ServerChart, you can bring a wide range of charts and graphs right to their Web browsers, giving them real-time access to business-critical data in an intuitive environment.

SILVERSTREAM
$495.00 Extend Application Server
Developer Edition (5 User)

SilverStream eXtend allows you to rapidly create, assemble, consume and deploy Web Services. It includes an advanced and intuitive development environment. This workbench provides everything for developers and business analysts to create, assemble, consume and deploy services. And it integrates with your favorite code editors and source control systems.

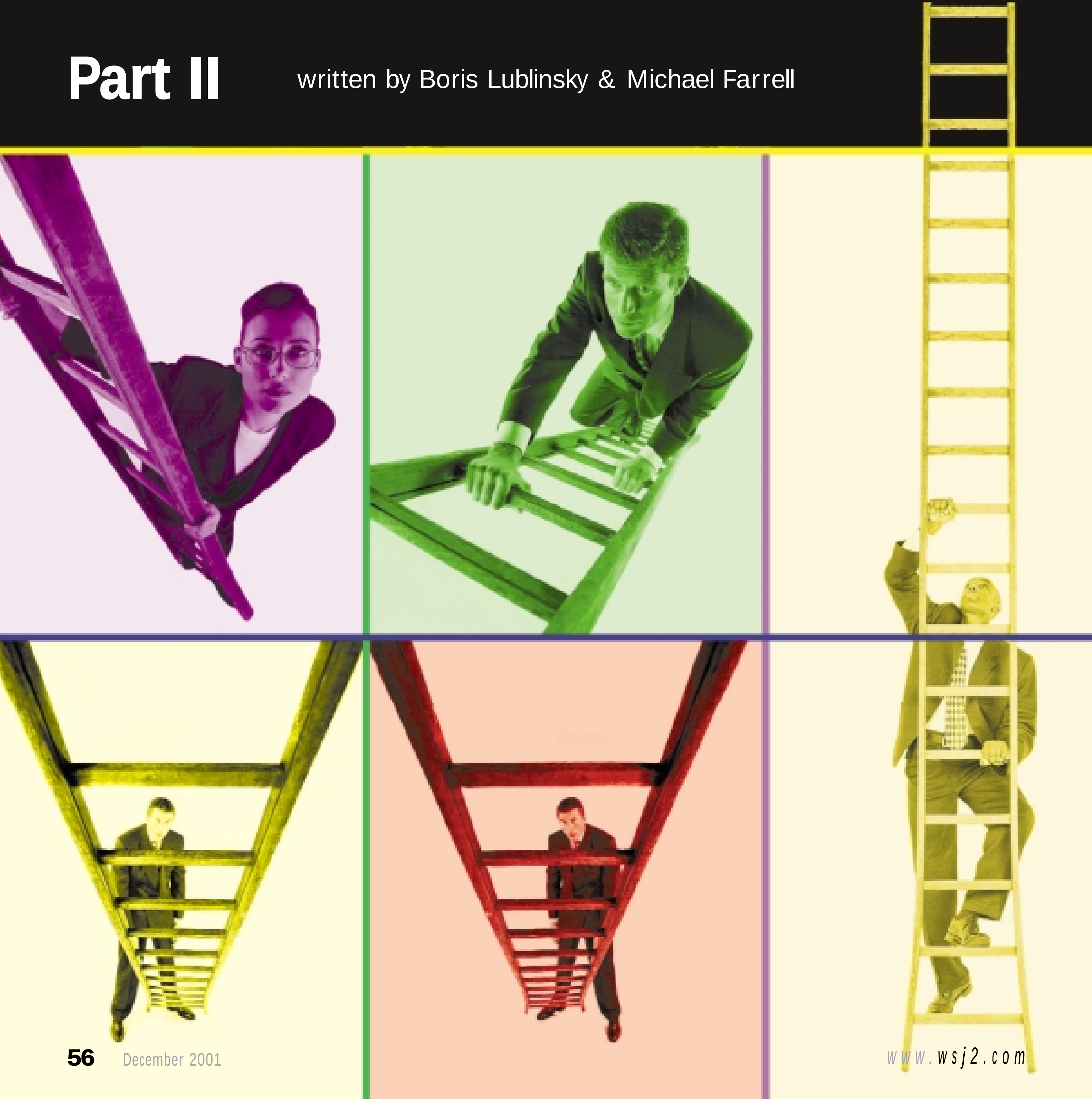
W W W . J D J S T O R E . C O M   OFFER SUBJECT TO CHANGE WITHOUT NOTICE

WSJ FEATURE

WEBSERVICES and Distributed Component Platforms

Part II

written by Boris Lublinsky & Michael Farrell

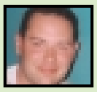


How They Stack Up

Author Bios

Boris Lublinsky is a regional director of technology at Inventa Technologies, 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

In part one of this article (***Web Services Journal***, Vol. 1, issue 1), we provided descriptions of what Web services actually are, including their foundation technologies—WSDL and UDDI—and discussed the fundamental characteristics of distributed architectures in general. Here we use these characteristics to compare Web services with other popular distributed architectures. Major business applications for Web services are also presented. We conclude with our analysis of the true readiness of Web services.

Comparison of Web Services to Other Distributed Systems Implementations

Based on the fundamental characteristics of distributed architectures described in part one of this article, Tables 1–3 compare Web services (based on SOAP binding) with other popular distributed systems implementations: CORBA, DCOM, and RMI. These tables were compiled based on the structuring discussed.

TABLE 1: Basic Principles

	Web Services	CORBA	DCOM	RMI
Request and Response	A client issues a request to execute a method of a service implementation. A server is always remote, either in a separate process or on a separate node, accessible over HTTP. Invocation is always synchronous. Server object always contains complete implementation.	A client issues a request to execute a method of an object implementation. There is no constraint imposed on the location of the client and the requested object implementation; they can share an address space, can be located in separate address spaces on the same node, or can be located on separate nodes. A server (process) can contain implementation of several objects or a single object, or even provide an implementation of one particular method only.	A client issues a request targeting a particular interface of a server object; the server object can reside in the same process as the client (inprocess server), in a separate process but on the same node as the client (local server), or on a separate node (remote server). In DCOM, remote method calls are synchronous.	In addition to the classical scenario of a client request targeting a particular interface of a single server object, the RMI architecture is open to support replication of server objects (transparent to the client request). The Sun Java RMI implementation currently includes only simple point-to-point requests and responses based on the underlying TCP-based streams.
Remote Reference	Web services do not define remote reference, which can be converted to a remote pointer. A string representation of the URL of the Web services location can be considered as a "limited" object reference.	Remote references are called Object References; the target of an Object Reference must be an object that supports the CORBA::Object IDL interface (such objects are CORBA objects).	Remote references are handles to server objects' interfaces. Each of the separate interfaces of a server object is associated with its own remote reference. Remote references are stored in proxy objects.	Remote references are handles to remote objects. Remote references are not directly accessible; they are encapsulated in proxies. A key RMI feature is that a remote reference can always be employed only via an interface. This conceptually separates the interfaces and implementations of objects.

TABLE 1: Basic Principles continued

	Web Services	CORBA	DCOM	RMI
IDL Interface	Web services are based on the WSDL IDL language. Both IBM and Microsoft provide WSDL compilers, allowing generation of server stubs and client's proxies.	CORBA specifies the CORBA IDL and its mapping to several programming languages (e.g., C++, Java, Smalltalk). The IDL language provides a means for interface definitions; there are no constructs related to object implementation.	The IDL language used by DCOM is called MIDL. Interface specifications are compiled by the Standard Microsoft IDL Compiler (also denoted as MIDL), which creates the code of server stubs and client proxies.	RMI uses the Java language constructs for interface specification (instead of employing a separate IDL language).
Object Proxy	Although it is feasible to use Web services without a client proxy, most of the implementations introduce stubs and proxies to make XML-based communications completely transparent to the user. This also allows for hiding the distribution aspect of Web services implementation.	The client-side proxy code is called an IDL stub; the server-side proxy code is referred to as an IDL skeleton. However, in CORBA the concept of “proxy” is used to denote an object created on the client side, which contains the IDL stub and provides some additional functionality, e.g. support for dynamic invocation.	Anytime a remote reference is provided to a client, a proxy object is automatically created (if it does not already exist) in the client’s process. The proxy object supports the same interface as the target of the remote reference. The client always works with a local reference to the proxy object.	Anytime a remote reference is provided to a client, a proxy is automatically created (if it does not already exist). The proxy supports the same remote interface as the target interface. The client always works with a local reference to the proxy.
Marshalling	In the Web services implementation, all marshalling is done via XML. Because object references are not supported by Web services, objects are always passed by value, using XML-based externalization.	Both request and response are delivered in the canonical format defined by the IIOP protocol, which is the base for CORBA interoperability. On the client side, marshalling is done in the IDL stub or via the functionality of the dynamic invocation interface. On the server side, unmarshalling is done in the IDL skeletons (or in the Dynamic Skeleton Interface, DSI) and partially in the Object Adapter. In a request (response), Object References can be provided as parameters; remote objects can be passed by reference or by value.	DCOM uses a proprietary internal format for request and response marshalling. As far as parameter passing of remote calls is concerned, data is marshalled in a platform-independent format called network data representation. When a reference to a server object’s interface is passed as a parameter the necessary stub and proxy objects are always created automatically.	RMI uses a proprietary internal format for request and response marshalling. As far as parameter passing in remote calls is concerned, local objects are passed by copy using Java object serialization. When a remote object is passed as a parameter (a reference to a proxy is indicated as the parameter) the proxy itself is passed (serialized).

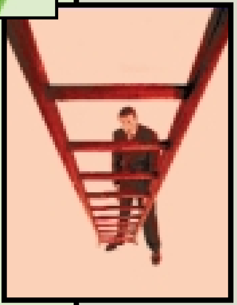
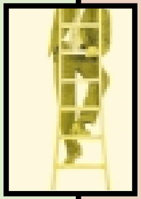


TABLE 2: Providing & Deploying Services

	Web Services	CORBA	DCOM	RMI
Registering Service with a Broker	Web services do not require a broker; thus, they do not support registration with a broker. They can only be registered with the corresponding UDDI.	Registering the object, which represents the service, is the task of the Object Adapter. When registering a new object, POA assigns a new Object Reference, which will serve for referencing the newly created object. Activation signals that a particular server (a particular POA) is ready to accept requests.	DCOM does not support Service registration with the broker. The basic strategy of working with remote objects is that the client asks a remote factory residing in the server for a remote object creation and at the same time indicates the interface used for accessing the object. During this process the corresponding stub and proxy objects are created.	A server object is registered together with all the remote interfaces it implements. The actual registration with the RMI broker is done via a special method. The result of the registration is a proxy supporting all of the remote interfaces the object implements.
Naming	UDDI can be used as a naming service for Web services.	The CORBA Naming Service is a standard CORBA service. However, in addition to the standard Naming Service, many CORBA implementations provide a vendor-specific way to locate objects associated directly with registering a service at the server side and a “binding” operation on the client side.	MIcrosoft is switching from a local, registry-based naming service to a global one based on the Active directory.	At every node supporting RMI, there is a daemon process called RMI Registry. In principle, an RMI Registry is a name server which supports flat name space and registers pairs <name, proxy> associated with the remote objects existing in this node.



Meet the distinguished experts in the Web Services world we're honored to have sitting on our Advisory Boards

WSJ

Who's on Board

Web Services Journal International Advisory Panel

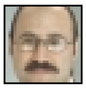
Andrew Astor has held technology leadership positions in diverse industries such as financial services, software development tools, and consulting organizations.

Steve Benfield is the CTO of SilverStream Software. He has more than 15 years of experience delivering innovative Web and IT solutions.

Dave Chappell is vice president and chief technology evangelist at Sonic Software and a member of the JAXM Expert Group. He is coauthor of *The Java Message Service* (O'Reilly & Associates).

Bob Crowley is president, CEO, and director of Bowstreet. He sits on the boards of directors for MetraTech, a next-generation billing company serving B2B utilizing Web services and XML; and Beachfire, a Web services platform provider of complex supply chain negotiation services.

Tyler Jewell is BEA's Principal Technology Evangelist. He is the coauthor of *Mastering Enterprise JavaBeans 2.0*, coauthor of *Professional Java Server Programming* (*J2EE 1.3*), is a regular J2EE columnist at www.onjava.com, and is the technology advisor to theserverside.com where he writes about Web services.


Paul Lipton is director of object technology at Computer Associates International, Inc. (CA). He is the CA technical representative to the ODMG (Object Data Management Group).

David Litwack is the CEO of SilverStream. In addition, he is a director of Trellix Corp., a Web site–creation software company; and of e-Room Technology, Inc., a provider of Internet-based software and services.

Anne Thomas Manes is the CTO of Systinet, a Web services infrastructure company. Anne is a recognized industry spokesperson and has published on a range of technology issues.

Norbert Mikula is a founding member of DataChannel. He serves as vice-chairman of the board of directors of OASIS and is industry editor of ***Web Services Journal***.

Graham Glass is the founder, CEO, and chief architect of The Mind Electric, which designs, builds, and licenses forward-thinking distributed computing infrastructure. He is the author of *Web Services*: *Building Blocks for Distributed Systems*.

Web Services Journal Technical Advisory Panel

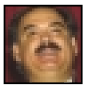
Dave Howard was an architect with ObjectSpace, Inc., and a founding member of the OpenBusiness team. He is an independent consultant in Dallas, TX.

JP Morgenthal is CTO of iKimbo. He is also author of *Enterprise Application Integration with XML and Java* and a world-renowned expert on distributed computing.

David Russell is CTO and cofounder of Bind Systems, a software vendor developing Web services technologies to enable collaborative electronic business.

Ajit Sagar is the founding editor and editor-in-chief of ***XML-Journal***. He is a senior solutions architect with VerticalNet Solutions.

Simeon Simeonov, chief architect at Allaire Corporation, is a member of the W3C working group on XML protocol and the J2EE expert groups on XML business messaging and XML data binding.


Dr. Richard Mark Soley is chairman and CEO of the Object Management Group (OMG), where he is ultimately responsible for all of its business, including board activities and oversight of the OMG's neutral and open technical process.

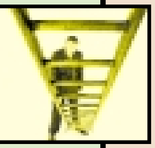
James Tauber is director, XML technology, at Bowstreet. He is a noted authority on XML and a principal editor of the emerging Directory Services Markup Language (DSML) for e-commerce.

TABLE 2: Providing & Deploying Services continued

	Web Services	CORBA	DCOM	RMI
Trading	UDDI can be used as a trading service for Web services.	CORBA trading service (analogous to Yellow Pages) provides a list of services (Object References) that possess certain properties, the value of which is used as the search key.	There is no trading or location service in DCOM at present.	There is no trading or location service in RMI at present.
Binding	Web services are implemented on top of connectionless HTTP protocol and do not require any binding.	The client is bound to the requested server after it receives one of its Object References (and a corresponding proxy is created).	The basic strategy of work with remote objects in DCOM is to ask a known class factory to create a server object and provide access to its particular interface.	Binding is done by accessing the RMI Registry and obtaining a registered proxy for a given interface.
Static Invocation	When the client knows the WSDL of the Web service and its exact location (URL), the remote call can be invoked statically.	When the client is compiled with knowledge of the requested service IDL specification, the proxy implements the mapped IDL interface methods. The corresponding code of the proxy is encapsulated in the IDL. Similarly, the corresponding IDL skeleton is the code of the server-side proxy, which implements static delivery.	When the client code is compiled with knowledge of the requested service interface, the remote calls can be encoded statically as calls of the proxy object's methods.	If the client was compiled with knowledge of the requested service interface, it can use the methods of the corresponding proxy via statically encoded calls. Similarly, the corresponding skeleton at the server-side implements static delivery.
Dynamic Invocation	Dynamic invocation in Web services is based on the information provided in the UDDI.	For dynamic invocations, the proxy contains a special method, which creates a request object. There is also a way to submit arguments with the request. When creating a request, the Interface Repository can be consulted for details about the target interface.	Dynamic invocation in DCOM is based on the IDispatch interface, which is required by any DCOM object. IDispatch allows for introspection of the remote interfaces.	There is no support for dynamic delivery on the server side. On the client side, however, dynamic invocation is always possible via the implicit opportunity to use Java introspection.

TABLE 3: Inherent Characteristics of the Distributed Architectures

	Web Services	CORBA	DCOM	RMI
Server Objects Garbage Collection	Because Web services are based on HTTP, which is a connectionless protocol, the life cycle of the Web services corresponds to the duration of the request; thus server objects garbage collection is not necessary.	CORBA addresses the garbage collection problem via reference counting.	Garbage collection in DCOM is based on reference counting and internal pinging, validating the fact that connections are alive.	Each registration of a remote reference in the RMI Registry implies one live connection. RMI addresses garbage collection based on the TCP/IP connections counting.
Transactions	There is no transactional support for Web services and introduction of one is not feasible. Execution of Web services is an independent transaction with it own commit brackets.	CORBA Transaction Service (OTS) defines CORBA support for distributed transactions, including two-phase commit.	Transactions are supported through the Microsoft Transaction Server (MTS).	RMI supports Java Transaction Service (JTS), which is a mapping of the CORBA Transaction Service to the Java environment.
Concurrency	Java implementations are built on a servlet engine, which is inherently multithreaded. Either Java synchronization mechanisms or a SingleThreadModel Servlet interface can be used to handle concurrent access. Microsoft's implementation is based on DCOM; thus, DCOM concurrency support can be used in this case.	In multithreaded servers, standard implementation (language-dependent) synchronization tools are supposed to be employed in the server object to avoid race conditions.	DCOM defines two basic threading models. The Apartment model is a single-threaded serializing request execution and no synchronization is required when it is used. A free threading model supports multithreading execution of the request and underlying OS. Synchronization mechanisms must be employed to handle concurrent access to data.	RMI system can deliver multiple calls at the same time to a particular server object and they will be executed in separate threads. The standard Java synchronization tools can be employed to handle concurrent access/modification to the server object's state.
Reactive Programming	There is no support for reactive programming in Web services.	CORBA Event Service supports reactive programming in CORBA. This service provides the event channel abstraction.	Reactive programming in DCOM is implemented through connection points and outgoing interfaces. A publisher announces a set of interfaces it will call to report on an event. A subscriber, sink object, implements some of the outgoing interfaces and subscribes with the particular connection point.	Java Beans event model can be used with RMI to provide reactive programming.



As we can see from the tables, Web services measure up quite nicely with other distributed systems environments. The main differentiating characteristics are:

- Web services, unlike other distributed systems, support only remote communications, they don't support colocated services.
- Web services are implemented over HTTP, which is a connectionless protocol. Therefore, some important features of distributed systems are not applicable for Web services. Those features include:
  - Remote references are not supported by Web services. A string-based URL, coupled with extra XML information, can serve as a remote reference.
  - Creation of the remote objects is not applicable to Web services. In Web services, a remote object is created by the underlying services for the duration of the request.
  - Synchronization of the life cycles of client and server objects (see above).
  - Since Web services are effectively stateless between invocations (see above), their usage must be based on a slightly different programming model. Every request/reply pair needs to be self-contained. This means that the request has to contain all of the information required for execution and the reply should return back all required information. The Web services programming model is closer to that of message-oriented middleware than programming models of distributed objects.
- Remote garbage collection in the case of Web services is not required.
- Web services use XML as a standard marshaling/unmarshalling mechanism (data representation). This allows their use without stubs/proxies when a client wants to create an XML-based request programmatically.
- UDDI plays a bigger role in Web services compared to that of the naming and trading services in other distributed environments. This is because Web services do not provide for creation of remote objects, and UDDI is the only way to obtain a "remote reference."
- Web services do not support distributed transactions. As such, building complex processes based on Web services requires the introduction of compensating transactions.
- Security and user authentication is a bigger issue for Web services, compared to other distributed architectures, since communications are happening over the public Internet and the number of nonregistered users, based on UDDI advertisement, is unpredictable.
- Web services do not provide event services, but implementing one similar to that of DCOM connection points is very simple.

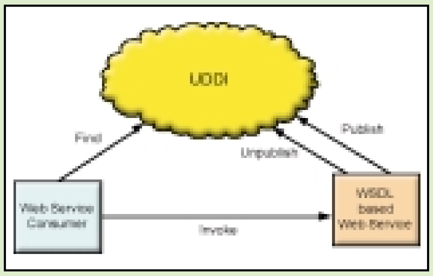


FIGURE 1 | Overall Architecture of Web services

The Promise of Web Services Architecture

Advantages of Web Services Architectures

The overall Web services architecture is shown in Figure 1. This architecture promises the following benefits:

- ***Better interoperability by minimizing the requirements for shared understanding:*** XML-based Web Services Description Language (WSDL) and a protocol of collaboration and negotiation (UDDI) are the only requirements for shared understanding between a service provider and a service requester. By limiting what is absolutely required for interoperability, collaborating Web services can be truly platform- and language-independent. These limited requirements mean that Web services can be implemented using a large number of different underlying infrastructures.
- ***Just-in-time integration:*** Collaborations in Web services are bound dynamically at run time. A service requester describes the capabilities of the service required and uses the service broker infrastructure to find an appropriate service. Once a service with the required capabilities is found, the information from the service's WSDL document is used to bind to it. Dynamic service discovery and invocation (publish, find, bind) and message-oriented collaboration yield applications with looser coupling, enabling just-

SAVE THE DATE! WIRELESS EDGE MAY 7-9, 2
BUSINESS & TECHNOLOGY
wireless
Going Wireless
WBT speaks with the president of AOL Anywhere

in-time integration of new applications and services. This in turn yields systems that are self-configuring, adaptive, and robust with fewer single points of failure.

- ***Reduced complexity by encapsulation:*** All components in Web services are services. What is important is the type of behavior a service provides, not how it is implemented. A WSDL document is the mechanism to describe the behavior encapsulated by a service. Encapsulation is key to:
  - *Coping with complexity*. System complexity is reduced when application designers are free from worrying about the implementation details of services they are invoking.
  - *Flexibility and scalability*. Substitution of different implementations of the same type of service, or multiple equivalent services, is possible at runtime.
  - *Extensibility*. Behavior is encapsulated and extended by providing new services with similar service descriptions.
- ***Better interoperability of legacy applications:*** By allowing legacy applications to be wrapped in WSDL and exposed as services, the Web services architecture easily enables new interoperability between these applications. In addition, security, middleware, and communications technologies can be wrapped to participate in a Web service as environmental prerequisites. Directory technologies, such as LDAP, can be wrapped to act as a service broker. By wrapping the underlying plumbing (communications layer, for example), services insulate the application programmer from the lower layers of the programming stack. This allows services to enable virtual enterprises to link their heterogeneous systems as required (through http-based communications) and/or to participate in single, administrative domain situations, where other communications mechanisms can provide a richer level of functionality.

Use of Web Services

Web services applications can be grouped into three major categories:

- ***Business Information***: Web services can be used to share information with consumers or other businesses. Examples of these types of services include weather reports, news feeds, airline schedules, stock quotes, and product catalogs.
- ***Business services***: Web services can provide transactional, fee-based services for customers. A global value of suppliers can be created that can be leveraged for conducting commerce. These types of services include auctions, marketplaces, credit checks, and ticket reservations.
- ***Business Process Externalization***: Web services allow for the aggregation of internal processes with those in another enterprise, thus creating business processes spanning multiple enterprises.

Business Information

Aggregating rich content by using Web services with business information is very similar to portals. An argument can be made to support favoring both technologies.

- Portals aggregate information statically from a fixed URL. They provide optimization technologies, like page caching, to minimize the number of HTTP requests. Modern portals have built-in security and personalization and frequently offer community services such as chat.
- Web services, combined with UDDI, allow for the dynamic aggregation of information. They can query UDDI Yellow Pages to find the most appropriate service, and then include content of this service in the presented information.

When Web services are used to provide rich content, data is returned in the form of either an HTML or XML string. In this case strong typing, supported by WSDL, is not really used.

Business Services

A service-oriented business founded on Web services is a promising proposition. This approach is very similar to the thin-client revolution. Instead of multiple installations and integrations with other enterprise systems, with the need to keep

WSJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
AltoWeb	www.altoweb.com		13
Apress	www.apress.com	212-533-3503	19
Attunity	www.attunity.com		11
BEA	www.developer.bea.com		2,3
DataMirror	www.datamirror.com/resourcecenter	800-362-5955	21
Infragistics	www.infragistics.com	800-231-8588	27
Java Developer's Journal	www.sys-con.com	800-513-7111	61
JDJ Store	www.jdjstore.com	888-303-JAVA	55
Shinka technologies	www.shinkatech.com/register7/	800-SHINKA1	23
SilverStream	www.silverstream.com		67
Software AG	www.sofrwareagusa.com	877-SAG-4XML ext. 133	9
Sonic Software	www.sonicsoftware.com		15
SpiritSoft	www.spiritsoft.net/downloads		4
SYS-CON Events	www.sys-con.com	201-802-3029	47
SYS-CON Media	www.sys-con.com	800-513-7111	39,55
Web Services Journal	www.sys-con.com	201-802-3024	33
WebSphere Developer's Journal	www.sys-con.com	800-513-7111	63
Wireless Edge Conference & Expo	www.sys-con.com	201-802-3069	65
XML Global Technologies	www.xmlglobal.com/newangle	800-201-1848	68
XML-Journal	www.sys-con.com	800-513-7111	64

Advertiser is fully responsible for all financial liability and terms of the contract executed by their agents or agencies who are acting on behalf of the advertiser.

track of distribution for maintenance and updates, software companies may host their products internally and provide their usage for a fee. This requires:
- Software companies to either have strong alliances with hosting companies or get in the hosting business themselves.
- A very high degree of reliability for these Web services.
- A very high degree of security for these Web services.
- A change in the company's attitude for outsourcing potentially business-critical applications.

A major advantage of this approach is the maintenance and support of applications, which are located in only one place. The disadvantages are:
- Inability of companies to customize applications in order to better fit their needs.
- Intermediate data in these applications, which might be useful for other applications within the company, is not readily available.

When implementing Web services for a service-oriented business, the use of UDDI is questionable. In this usage of Web services, we see establishment of upfront agreements between the service provider and the service user. It is highly unlikely that the service user will shop around through UDDI to pick the most appropriate Web services. The only reasonable play for UDDI in this case is to adjust a call to the possible interface change. Another issue here is request/response data. We foresee service-oriented business founded on Web services to be rather large chunks of execution accepting and returning a large amount of data. XML, in our opinion, is better used for data transfer in the strongly typed IDL, supported by WSDL. Using XML for this purpose allows for a more loosely coupled implementation.

Business Process Externalization

Business process externalization and b2b integration are some of the most prominent requirements for today's enterprise. This area is dominated by B2B server products. These tools provide communications over HTTP, guaranteed delivery features, XML parsing support, back-end systems integration, process automation, and more. Web services can have a very strong role in the actual message delivery, coupled with back-end systems integration. Back-end systems can be wrapped into Web services so that they accept messages over the Web. In reality, this does not solve the whole problem of true B2B integration. Web services may serve as an ideal implementation of public business processes, however, private business processes must be implemented on both sides of the firewall. In this case, as in the previous one, use of UDDI is questionable. Key business relationships typically remain intact for long periods of time. Also, we see mostly XML-based communications in this case, adhering to one of the available XML standards (cXML, xCBL, etc), rather than IDL-based communications.

Conclusion

Web services is a great proposition, and may make an excellent addition to the established distributed component platforms, but there is a long way to go to make it a reality for use in mission-critical business applications. As the Web services framework continues to solidify, here are a few things to keep in mind:
- In order for them to really take off, the reliability of Web service and UDDI server implementations has to increase dramatically. These machines have to be up and running all the time.
- Another prerequisite for widespread adoption of Web services is the complete "Webification" of computers. Users can connect to Web services only if they are on the Web. Today, many people still use their computers as disconnected boxes.
- Absence of distributed transaction support presents a serious obstacle for implementing mission-critical business processes based on Web services.
- The UDDI specification as it stands today does not guarantee proper taxonomy of Web services for Yellow Pages. This is left for the implementer to complete. Choice of the proper, universally accepted criteria here is crucial for the viability of UDDI.

wirelessEDGE conference&expo

Plan to Exhibit

Provide the Resources To Implement Wireless Strategy

The conference will motivate and educate. The expo is where attendees will want to turn ideas into reality. Be ready to offer solutions.

INTERNATIONAL WIRELESS BUSINESS & TECHNOLOGY CONFERENCE & EXPO

CONFERENCE & EXPO

CONFERENCE & EXPO

Shaping Wireless Strategy for the Enterprise

Santa Clara, CA | **May 7-9, 2002**

WirelessEdge will provide the depth and breadth of education and product resources to allow companies to shape and implement their wireless strategy. Developers, i-technology professionals and IT/IS management will eagerly attend.

Plan to Attend the 3-DAY Conference

WHO SHOULD ATTEND

Mobile & Wireless Application Professionals who are driving their enterprise's wireless initiatives:

- Program Developers
- Development Managers
- Project Managers
- Project Leaders
- Network Managers
- Senior IT and Business Executives

Conference Tracks

Track One: Development	Track Two: Connectivity	Track Three: Wireless Apps	Track Four: Hardware	Track Five: Business Futures
WAP	Smart Cards	Education	Cell Phones/	Wireless in Vertical Industries
i-Mode	Wireless LANs incl. Bluetooth	Health Care	WorldPhones	The WWWW
Bluetooth / 802.11	UMTS/3G Networks	Entertainment	PDAs	Unwired Management
Short Messaging	Satellite Broadband	Transport	Headphones/	From 3W to 4W: Issues and Trends
Interactive Gaming		Financial Services	Keyboards /	"Always-On" Management
GPS / Location-Based		Supply Chain Management	Peripherals	Exploiting the Bandwidth Edge
Wireless Java			Transmitters/	Unplugged Valueware
XML & Wireless Technologies			Base Stations	Wireless Sales & Marketing
			Tablets	

FOR INFORMATION CALL

201 802-3069

SPEAKER PROPOSALS INVITED

WWW.SYS-CON.COM

SHAPE YOUR WIRELESS STRATEGY SAVE THE DATES!

EXCLUSIVE SPONSORSHIPS AVAILABLE

Rise above the noise. Establish your company as a market leader. Deliver your message with the marketing support of

SYS-CON MEDIA

SYS-CON EVENTS

THE LARGEST WEST COAST WIRELESS

BUSINESS & TECHNOLOGY CONFERENCE OF THE YEAR!

Beyond Scratching the Surface

WRITTEN BY

Colin Evans

Colin Evans is director of e-Business Solutions Lab at Intel Corporation's Intel Architecture Group. He's responsible for the development of solutions architecture and technology that support Intel's e-business strategy and the development of Intel's open e-business standards. He chairs the executive board of RosettaNet, a multicompany initiative focused on building Internet-based supply chains for information technology, electronic components, and the semiconductor manufacturing industries. He also chairs the board of directors at OASIS, an international consortium driving XML interoperability standards.

COLIN.EVANS@INTEL.COM

To understand what all the fuss about Web services is, it's useful to start with a little history. As Forrest Gump's mother said, "You can't look forward without putting the past behind you."

So what have we all been doing this past decade?

First, we've made massive investments in systems that automate our company business processes. Sometimes we have connected them internally and sometimes we've connected them externally to our trading partners; usually we haven't done much of either. It's just too hard to justify and build the custom connections required to integrate all systems given the mosaic of data and technology and processes that have to be joined. So we've taken the approach, "Hook up the important ones; we'll tackle the rest later."

Second, we've obsessively embraced and exploited to an extraordinary degree the World Wide Web as an external layer to make our companies more attractive and accessible to our customers, suppliers, and employees. Despite the rise of "eSkepticism," nobody can dispute that this has changed forever all our expectations for the way we work together.

The rate of expansion of the Web has, however, definitely hit some natural speed bumps as the business environment has become more complex and business conditions have become tougher, with shorter product life cycles, pressure on margins, more outsourcing, more mergers, and more need for flexibility to compensate for world uncertainty.

At bottom, we are neither getting the most out of our legacy application investments nor is a browser-oriented Web taking full advantage of the power of available rich personal computers or the people who use them.

Looking Forward

Web services make discrete pieces of business logic and information programmatically available throughout the Web. Our personal computers become valuable consumers and integrators of these services and, just as the Web allows small businesses to project scale and global reach way beyond their size, these businesses will be able to assemble impressive capabilities from a few generic services and a laser focus on adding their own unique competence.

Knowledge and expertise can be exposed as a Web service – and there's no reason why every connectable computing device we own could not become a powerful consumer and useful producer of Web services. Following Metcalf's Law, the total value of the Web will grow exponentially with this increase in connections. So what is needed to unleash this potential?

From Web Sites to Web Services

The first variable of this enormous potential is standards. Many companies managed to create useful information-management and business tools on the Internet long before we all learned what *WWW* stood for, but the Web's expansion moved at warp speed only when it became possible for nontechnical users (my Mum and Dad) and relatively unsophisticated developers (me) to simply describe, locate, and connect to Web sites. This was all made possible, of course, by the ubiquity of hardware and software that implemented HTML, DNS, and HTTP: a standards-enabled revolution.

Now, as we have all grabbed XML as a way to create smart data, the equivalent early adoption is just starting. The real explosion, however, happens when it is again trivially easy to describe, locate, and connect to Web services. This is the promise of WSDL, UDDI, and SOAP, which are receiving unprecedented acceptance from all major software protagonists – including Intel.

After standards, we need tools. With an open architecture based on a few critical standards, it's possible to build tools that allow us to build, publish, and consume Web services using both the company's central computing resources and our own personal computing devices – at work, at home, in the office, on the road, when we're there, and when we're asleep.

With the right tools, the following scenarios will become commonplace:

- My home computer will pull financial information from my broker, my bank, and my employer into my money manager and let me know if I'm on my personal budget or spending like a drunken sailor.
- A buyer's laptop can keep track of stock and lead-time information in the factory as well as at suppliers, and then notify them of critical situations requiring individual intervention.
- A large insurance company can make its products accessible as a Web service that can be assembled into offers by banks, travel companies, car dealers.
- The three companies you just acquired can expose internally their financial and factory data as Web services so you can feed decision support and accounting systems without converting everyone to the corporate ERP system.

Real-time on a Global Scale

Through Web services and their extension into peer services, we can support these ad hoc interactions in real-time and directly between individuals – whether subscribers are on another floor of the building or across the world. New intuitive, highly empowering tools for line-of-business users will allow them to assemble, at their personal desktops or mobile PCs, their own views of the world by aggregation of other peoples' and companies' published Web services and make it possible to implement new business relationships and processes virtually on the fly.

This is the realization of the true promise of the Web; the e-business Internet overflowing with opportunities to invent new ways to do business, increase productivity, and generate growth.

Silverstream

www.silverstream.com

XML Global

www.xmlglobal.com/newangle