THE NEW PROGRAMMING PARADIGM???

November 2001 Volume1 Issue 2

WebServices JOURNAL

WSJ2.COM

FULL CONFERENCE COVERAGE

web services EDGE conference&expo

pg. 44

SEPT. 23–26 IN NYC

A Web Services Success Story

DUPONT PROTECTIVE COATINGS

pg.14 by Catherine Marchand

SYS-CON MEDIA

BEA

www.

WebServices JOURNAL
.NET J2EE XML

INTERNATIONAL ADVISORY BOARD
JEREMY ALLAIRE, ANDREW ASTOR, STEVE BENFIELD, DAVE CHAPPELL, BOB CROWLEY, GRAHAM GLASS, TYLER JEWELL, 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
BRIAN BARBASH, STEVE BENFIELD, BEN BERNHARD, KURT CAGLE, DAVE CHAPPELL, NOEL CLARKE, REBECCA DIAS, DARREN GOVONI, ANDY GROVE, PHIL KARECKI, UNA KEARNS, CATHY MARCHAND, NORBERT MIKULA, SEAN RHODY, DAVID RUSSELL

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
ASSISTANT CONTROLLER: JUDITH CALNAN
ACCOUNTS PAYABLE: JAN BRAIDECH
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
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, PURVA DAVE, CAROL AUSLANDER
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

The Information Paradox

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

One of the most interesting paradoxes of the information age is the challenge of obtaining critical mass for a technology – the classic chicken-and-egg problem. Remember when you could buy your software on floppy disks because only a few people had CD-ROM drives? How about the plethora of high-density floppy drive replacements? Our industry has frequent problems adapting technology, in part because there are so many players, and so few who will work together. It's interesting to see that many folks regard Web services as yet another possible technology, but one without widespread adoption.

Interesting, but only partially true. It's definitely true that Web services are in their early stages – standards are being proposed, emerging, failing, and competing. Vendors are moving quickly to carve out their turf along the usual battle lines. There's Microsoft, of course, with its .NET vision and grand theories of being everyone's electronic wallet, and taking a small piece of every electronic transaction. Then there's the Open Standards group – Sun, IBM, HP, BEA, and the like–that champions a more utopian solution where transaction fees are left to individual vendors, and whose software provides the interconnectivity.

What most people are quick to point out is that some of the enabling technologies proposed for Web services are either on the drawing board, in committee, or have significant implementation challenges.

But what's ignored by those who want to keep Web services in a fluid state in order to seize the ideological and financial high ground is the fact that some forms of Web services are already widespread.

In our first issue we looked at Dun & Bradstreet's Global Access Toolkit, which allows customers to access the wealth of financial statistics that D&B provides as part of its own business processes. In this issue we discuss how Dupont is changing its business climate with Web services. Future issues will cover other well-known companies that are moving along with Web services.

Clearly, not all of these services meet the most narrow of definitions of Web services. I've had discussions with some technology companies that want to define Web services along very narrow, strictly technological boundaries. If it doesn't use UDDI, it's just not a Web service. If it's not a centralized server, it must be something else. These companies will undoubtedly be disappointed in my definition of a Web service, but there's a strong reason for adopting a broader view – adoption of Web services will be a business decision, not a technology decision.

If we've learned one thing from the dot-coms, it's that technology for technology's sake is a sure path to failure. It wasn't the Internet that was the revolution; it was the change in business models that resulted from increased connectivity that was the true revolutionary force. Sites like eBay aren't distinguished by technology – they're distinguished by business principles and superior service. Let's face it, almost all of the dot-com companies used the same technologies. And the brick and mortars cleaned their clocks, using the same technologies but providing better service. The myriad of Net markets showed the various industries that spot markets could in fact contribute value – but their business propositions had no underlying staying power because any large company with enough funding could create their own Net market and use their liquidity and established partnerships to drive the traffic to their site instead.

The same issue applies to Web services, and really to any new technological solution. There needs to be a business value for a technology, or there will be no adoption. A strict adherence to definitions of technology doesn't provide that business value. It doesn't matter if the service uses all of the components of the technology as long as it embodies the spirit of the technology. D&B provides information worldwide using an XML interface and some of the other components of the technology but has not implemented UDDI or WSDL. It's still a Web service.

In coming issues we'll focus on a variety of the technologies that will make Web services successful. Some, like UDDI, SOAP, ebXML, and WSDL, fit almost everyone's definition of a Web service. And that's good, because we have a common denominator of significant capabilities. But we'll also explore other aspects of the technology where definitions diverge – things like Business Process Management, Workflow Definition Language, and peer-to-peer computing. And we'll focus on business issues as well as technology, because business reasons will drive the adoption of Web services.

It's likely that you're reading this because you want to know about Web services – what they are, how to do them, what they can do for your business. And you've come to the right place. We're here to answer those questions. Stay tuned as we watch Web services evolve and grow.

written by Una Kearns

Content Management & Web Services

eContent Services– the services behind Web Services

Una Kearns, XML architect at Documentum, leads the development of Documentum's next-generation XML solutions and is responsible for the company's overall XML product strategy and architecture. She is a member of the OASIS board of directors and XML.ORG advisory committee, and is the Documentum representative to UDDI.org and the W3C. She is also an active participant and spokesperson in industry events and conferences.
QUESTIONS@DOCUMENTUM.COM

Over the course of the last year, we've all read many articles on Web services and the various standards forming to address the different levels of the Web services stack (SOAP, UDDI, WSDL, ebXML). The stock quote example is becoming as recognized as "Hello World" from our introductory programming books. From responses to a recent poll in an online magazine, it is evident that there is still confusion over what Web services actually are and how they can be used effectively by organizations. One of the main reasons for this is that Web services are only as good as the service they offer, and that the benefits of Web services often depend on the domain within which they are applied.

This is understandable when you consider that at a basic level Web services are all about exposing, publishing, and invoking a programmatic interface to a business application service. A service that lists addresses and ratings of restaurants within a given radius may be very useful for a travel portal, but it is not useful for telecommunication's IT department building a contract management application for managing service level agreements across network providers. These examples also highlight the different types of service interactions that may be involved in a Web services environment, from intraenterprise and interenterprise to business-to-consumer, and from simple to complex, such as services that exist as part of an entire business process (e.g., contract negotiation).

In this article we will examine how Web services can be used in the domain of content management.

Why content management? Applications of content management illustrate the various types and usage model benefits associated with the new Web services model. We have heard much discussion around the application server market in relation to Web services (the new battle after J2EE), but we have heard little about the other components of the e-business stack, which are often at the heart of applications business logic.

Before we go into the details of Web services and content management, let's recap briefly some of the core concepts of the Web services model and the role of content and content management in e-business applications.

Content-Rich Applications

So what does content have to do with e-business? Quite a lot when you think about it – much of e-business is based around the exchange of information. How could you buy or sell a product without up-to-date product information or a catalog?

The increasing importance of content management as a critical e-business infrastructure component has been driven by the explosive growth in the use of the Internet over the last few years. Until recently, organizations have mainly been concerned with managing their structured data, and gave little thought to how unstructured content assets were managed and controlled. As the use of the Internet increased, and Web sites both internal and external grew in complexity and size, streamlining the aggregation, creation, management, and distribution of content assets became critical for organizations to effectively capitalize on the Internet and move their businesses online.

Today, a Web content management application is vital for organizations moving online, but with the evolution of e-business systems, the management of all content assets becomes crucial. The word *content* itself is often misleading, as *content* represents core business assets from contracts and catalogs to product information, RFPs, and even the code that makes a Web site function.

Similar to the last decade in the data-centric world, where the RDBMS started out managing data and evolved to managing higher level chains of business applications, now content-centric applications such as contract management, RFP management, SOP management, catalog management, and so forth, have emerged. These applications are indispensable to companies as they fully integrate and automate their business processes.

An enterprise e-business infrastructure is built from a stack of best-of-breed products such as enterprise information portals, application servers, databases, and enterprise applications (ERP, CRM). The content platform resides as part of a full ecosystem of products. As with other business applications, these content applications are required to seamlessly integrate with both internal and external applications, be delivered through multiple interfaces (portals, Web sites), and, more interestingly, are often based around rich content, which merges content and data to produce a unified view of information. For example, consider a contract, which combines customer information, pricing information, and legal text, or a product catalog, which combines all types of content assets from images and data specifications to SKUs and pricing information.

Now, let's take a closer look at Web services…

The Next Stage in e-Business Evolution

Web services represent an important next step in how we use the Internet, and may

revolutionize not only how we think about application development and deployment, but how we do business.

To the extent of creating many new business models, we have seen the impact on every industry of the move from mainframe to client/server to the Internet. However, until recently the use of the Internet was largely concerned with human-based interaction through browsers. The back-end integration between business systems involved a lot of work, never mind considering when this integration occurs across enterprises. There are two reasons for this. Applications are generally developed in isolation from how they need to interact with other applications and systems. And integration generally involves a lot of "hard wiring," a busload of system integrators, and plenty of money.

In today's economy, companies are facing conflicting requirements – a shortage of IT budget but a pressing need to automate their business processes to remain competitive and drive down costs. Due to the ever-changing nature of business, partnerships, and technology, business systems need to be agile and adaptable. They need to integrate with each other across the extended enterprise regardless of the language they are written in, the platform they run on, and where they run.

In the next stage in the evolution of e-business systems, applications will be built from a set of service components that can be published, discovered, combined, and consumed over the Internet. The eventual goal is applications that can be dynamically assembled according to changing business needs, and personalized based on device and user access. In essence, services are self-contained building blocks that can act in a plug-and-play manner with other services to provide business functions that run on any networked platform.

From a technology and investment standpoint, this next stage is an evolution, not a revolution. In today's economy, companies are loath to discard existing investments in their legacy business systems.

The Web services model represents a merging of the distributed systems model with Internet computing. Applications can now be built, and dynamically combined, in a loosely coupled manner using Web services in a distributed networked environment. Web services expose self-contained business functions that are the result of an evolution in programming and application development from object-oriented and component-based models to a services-based model. The move to a service model is distinguished from the traditional component model by the addition of the notion of a service contract. The service contract is what enables complete separation of how and where the service is invoked from how the service is implemented. A service becomes a Web service when it can be discovered and invoked over the Internet. Discovery is based on it being published – in the most basic case in a manual, but most likely in a service registry (e.g., UDDI).

Currently, a Web services contract is defined using WSDL; published in a service registry (UDDI); and invoked over HTTP using SOAP, with XML being the neutral glue. Services can be implemented in any programming language (EJBs, Java Classes, COM, COBOL, C) and can be deployed on any platform (Windows, UNIX, mainframe) as long as it is network accessible.

The real power of services can be realized by the ability to chain them together to build applications that fulfill complex business processes without the need for hundreds of programmers, but rather by business users, or even better dynamically, which is the ultimate goal.

Usage Scenarios

As we have seen, content management is a broad area that deals with the aggregation, creation, management, and distribution of content assets. Content flows across systems and enterprise boundaries (see Figure 1).

The various applications of content management involve the combination of business logic and content. There are four main roles for using Web services with content management:

- Exposing and publishing content management services as Web services
- Invoking Web services as part of content management applications
- Extending business processes intra- and interenterprise
- Managing Web services

For each of these roles, Web services may be accessed at multiple levels, either within an enterprise, across the extended enterprise with partners and suppliers, or as generally accessible Web services over the Internet. Generally, most enterprises will not freely publish their core content management services over the Internet unless this is their actual business model, but will leverage services to rapidly integrate their applications across enterprise systems, and extend functionality to their extended enterprise (customers, partners, suppliers) where appropriate. As Web services become more the

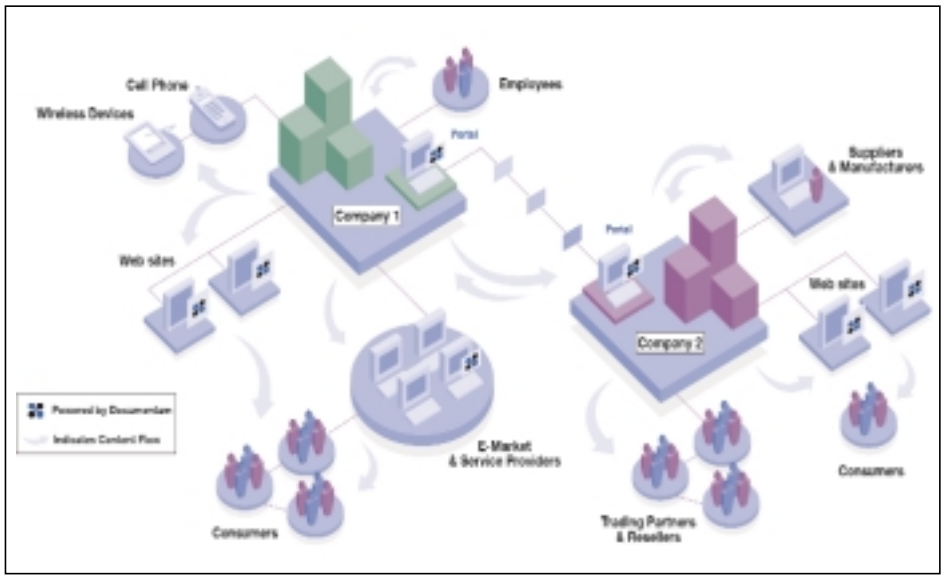


FIGURE 1 | Content flow across systems

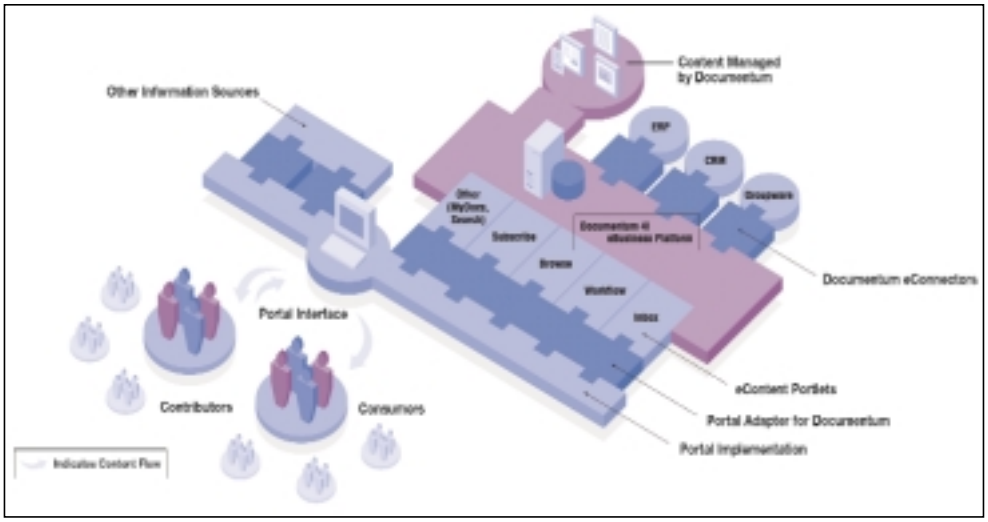


FIGURE 2 | **Content services for portals**

norm for software deployment and services (e.g., pay-per-usage models), corporations will begin to leverage external services for components of their applications.

Exposing Content Management Components

Business logic encapsulated in core components can be exposed as Web services to provide access through other applications (portals, enterprise applications, and external systems) and to rapidly build new applications by leveraging core services of the content management system – reducing overall costs and deployment time for organizations.

Web services will become a natural mechanism for integrating content management across systems, e.g., CRM, ERP, and portals. Vendors of these products will generally provide the integrations, which are key across the e-business stack, and companies nowadays are pushing for tighter, more complete integration out of the box.

The portal is one very good example. The enterprise portal is becoming the WebTop of the enterprise. Companies no longer want to deal with multiple application front-end interfaces for their employees. Content management is central to a portal in two respects:

1. Enabling employees, partners, and customers to participate in content management applications through one seamless interface
2. Powering portals with accurate content

Exposing content management components as services enables portal builders to rapidly include content management capabilities in the various enterprise portals.

Documentum, for example, has already released a set of services for portals (eContent services or portlets) that allow for core content management components such as inbox for workflow, subscriptions, and browse to be easily incorporated into any portal.

Exposing these portlets as Web services allows for more intelligence within the portal. Instead of having multiple disconnected components, the components can interact through the published interface. For example, for a customer service portal, a sales support person may look up a customer name through a portlet into the CRM system, while the browse portlet to the content management system automatically updates the contract view to show contracts just for that customer.

Developer and application builders will also leverage core services of the content management platform to build content management applications more rapidly, and will develop their own applications as services that their customers can readily use within the enterprise.

For example, a number of companies develop content management– based applications using Documentum as the foundation, e.g. contract and catalog management systems. These vendors will use the core services of the platform to rapidly build and get their products and ser-vices to market faster. They may want to expose their applications as higher level services to include, for example, contract management as part of an e-procurement system, or for market site providers to easily add catalog management and aggregation for their suppliers.

Corporations will also build Web services to expose their in-house applications both within the enterprise and to external partners. Exposing services will be done using standard interfaces (e.g., RossettaNet PIPs) or by agreed-upon interfaces between partners. One of the

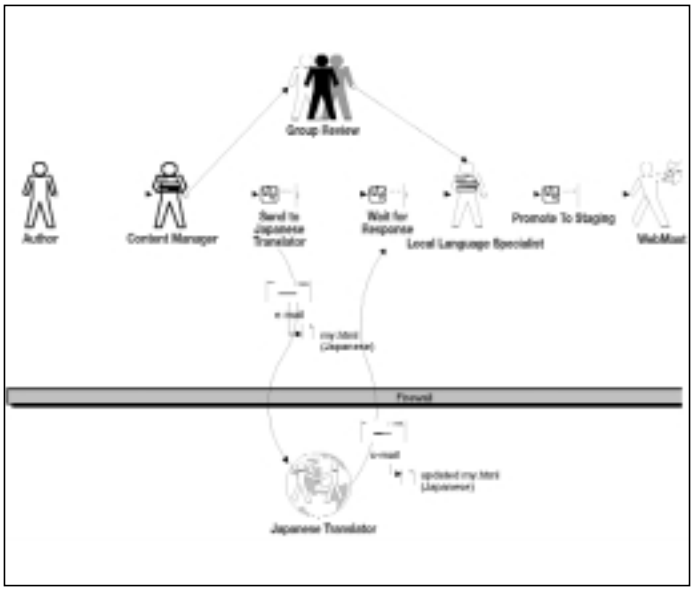


FIGURE 3 | **Extending business processes**

PIPS (2A9) is a good example of a content-based service that would be exposed by a corporation to their partners. This PIP is used in the semi-conductor industry and allows for companies to search across partners for information on available components that meet certain criteria. Another example might be a financial services company that would like to provide a new offering to some of its value-added customers by providing access

from their portal to certain subscribed research reports.

Invoking Web Services

Content management applications can also invoke Web services to extend functionality. For example, Documentum has many third-party technology providers who provide complimentary technology, e.g., translation software, transformation technology. These third parties will be able to expose and publish their components as services so they can be easily plugged into customers' systems either run remotely by the third party or installed at the customer's site. One example could be transformation. Different third parties might offer services for XML to PDF transformation, GIF to JPEG, or even more complex transformation services such as from one catalog schema to another, e.g., RosettaNet to XCBL. Customers could either search for companies that offer these services using a service registry or they could tell the system that they would like this particular document transformed from X to Y and it would automatically find a service that offers this and invoke the transformation.

Extending Business Processes

The business process/workflow around content can be extended to support external sources, e.g., negotiation of contracts, as is common with many applications, and participate in workflow spanning systems and enterprises (see Figure 3).

Managing Web Services

Earlier, we looked at examples of how services can benefit the entire eco-system of content management, but there is also an entire area that deals with how services are created, managed, and deployed.

Web content management has been one of the main applications that has allowed organizations to effectively manage all the content and code that supports their Web site infrastructure. As we now move to the next generation of the Internet, from using it primarily as a mechanism for human interaction to one of machine-to-machine or system-to-system interaction, there will be a need for companies to effectively create, manage, and deploy services. Web content management systems will evolve to fill this need, providing for the versioning and management of services, and the deployment of these service descriptors to registries so they can be accessed.

Content Management Platform Requirements

A scalable, open content management platform is required as the foundation of content-rich e-business applications that enable the successful and rapid integration of content across the entire information chain. The content management platform you select must offer a comprehensive set of services to deal with all aspects of managing your content assets on a global scale. More importantly, this platform must embrace and support the core technology foundations of the next-generation Internet (portals, J2EE, .NET, Web services, XML), allowing rapid development and deployment and easy integration with all business applications.

How Long Will It Take?

We may all have thought that the development of the Internet and the move to the browser happened extremely quickly, but the Web services model is likely to be adopted at an even more rapid pace. There are a number of reasons
for this:

- All major platform and infrastructure providers are quickly adding support for the Web services model into their platform offerings.
- Web services are not a revolution but an evolution of investments already made.
- They are simple in concept and leverage existing implementations – the same reasons the Internet was so popular initially.
- They will solve real problems of interoperability across systems and integration of systems components, allowing rapid application development and, eventually, dynamic application development.
- They will offer a substantial competitive advantage through the realization of operational efficiencies from automation across the extended enterprise and the reduction of the huge integration costs incurred by the current model of application integration. These benefits alone will be very appealing to organizations in the current economic climate.

Apart from the benefits of Web services already described, they also offer other potential opportunities, for example:

- New business models for software delivery: Pay-per-usage models, pay-per-service, and realization of hosted business models
- Unparalleled scalability using the Internet as the network and all connected devices as the computing power (assuming network bandwidth issues resolved)

Not everything will be exposed as Web services, but Web services do offer a large number of benefits to content management applications. There is still much work to do and obviously the more that is available as Web services, the more value a company is going to get from this model.

Altoweb

www.

Written by Catherine Marchand

Connecting to Customers with Web Portals

DuPont unit finds the right chemistry by moving online

Most companies find it easy to recognize the need to move their business online. Customers call for more timely information and easier access to online ordering as well as training and documentation. Internal operations and sales people want to leverage the cost benefits and efficiencies of electronically connecting to customers. But the real challenge is determining what to do and how to go about achieving it. DuPont Performance Coatings (DPC), a strategic business unit of DuPont, found itself in this position as it began considering the efforts to move its business online.

DuPont Performance Coatings was formed in March of 1999 by the combination of Herberts GmbH and DuPont Automotive Finishes, creating the largest automotive coatings company in the world. The company is divided into five business units: DuPont Herberts Automotive Systems, the worldwide largest manufacturer of automotive OEM coatings which sells directly to large manufacturers; DuPont Industrial Coatings, the leading supplier in selected markets in mass-production, industrial coatings; DuPont Ink Jet, the leading supplier of printing inks to companies like Hewlett-Packard; DuPont Powder Coatings, the second-largest powder coatings manufacturer worldwide; and DPC Refinish Systems, the largest manufacturer worldwide of coatings for car repair. DuPont Performance Coatings consists of numerous world-renowned brands, including DuPont Finishes, Standox, Spies Hecker, Nason, and DuPont Industrial Coatings. With more than 12,000 employees in over 35 countries in all continents, DuPont Performance Coatings is a global company with products and services in almost every market.

With no broad-reaching e-business plan, DuPont Performance Coatings began its e-business initiative by going to its customers, distributors, and partners to research what would take it beyond cost cutting and improvement of internal efficiencies. Ultimately, DPC wanted to use its Web-based e-business to create a competitive advantage and provide new value to these constituents.

DPC's management focused the effort on the needs of their customers but also remained vigilant in creating value for DPC's internal users. They didn't want the effort to be just a competitive "catching up" or simple problem-fixing, but a new method of coming to the market, communicating with customers, and potentially creating new business models.

Meeting Challenges

DPC decided to focus its first Web portal deployments on its automotive refinishing customers. In the U.S., this market consists of some 4,000 paint distributors and over 60,000 body shops. Each has particular specialties, brand preferences, and expertise requiring customized information and services. As DPC scoped out its e-commerce initiative, it rapidly became clear that bringing its ideal e-business solution to market would face considerable technical challenges.

The first big challenge was DPC's desire to build a global foundation to bring all the business units and brands together on a single technology and a single platform. After DPC's early e-business assessment, it was clear that bringing these disparate initiatives together into a Web portal was the most efficient strategy.

DPC wanted a long life expectancy for the solution. The solution needed to be able to support all of DPC's future global initiatives, so it was critical that the solution not be obsolete the day it was turned on.

DPC believed strongly that mass customization of these portals (tuned to the needs of each customer, distributor, and partner, and brand-targeted and customized for the information needs of each) would provide competitive differentiation. Closely related to this need is that the solution needed to support multiple business and marketing strategies. While the initial portal releases were targeted at particular markets and business units, the long-term goal of having a single platform required it to be flexible and adaptable to a variety of concurrent and unique business models.

DPC's solution had to support global privacy and legal requirements, as well as its own internal requirements for security and user access. Security concerns frequently rank high in e-business initiatives and DPC was no exception.

The final major challenge DPC faced was to find a solution that could be managed from a business standpoint rather than from a programming standpoint. DPC's goal was to have people in the sales organization who were responsible for the different markets and customers, and the different functional area managers, to each have control of content, input, and customer view decisions without having to rely on developers or other programming resources.

Designing a Strategy

The other possible solutions that were evaluated quickly proved impractical for the kind of unique portal views DPC wanted to offer its distributors and body shops. An ever-growing and increasingly complex code base would result in application performance issues down the

Author Bio

Catherine Marchand is an e-Business strategy manager with DuPont. In her 22 years with that firm, she has held positions in research and development, color and product development, marketing, and business, all in support of the automotive collision industry. She holds a BS in chemistry from the University of Michigan. CATHERINE.A.MARCHAND@USA.DUPONT.COM

road. Additionally, as the code base grew, more hardware would be required to deliver acceptable levels of performance. Plus, DPC would be forced to employ an ongoing team of highly skilled programmers to maintain the portal infrastructure.

The DPC e-Business Initiative team, with the advice of Computer Sciences Corporation's (CSC) NetBusiness Practice (e-commerce solutions for F500 companies), charted a strategy based on XML Web services and the Bowstreet Business Web Factory. DPC chose Bowstreet's Business Web Factory platform because it enabled them to:

- Decrease costs by eliminating the need to custom code, or hardwire, each new partner or customer to the portal
- Build customizable portals quickly and cost-effectively to respond to competitors' e-business initiatives
- Scale to include all of DPC's distributors and body shops worldwide, as well as all of the distributors and customers of its partners and competitors
- Avoid the scalability issues present in other vendor's solutions
- Properly capture and market the branding characteristics of each of the paint brands marketed by DPC

Construction of the portals got underway late in September 2001, led by Bowstreet professional services personnel and consultants from CSC. Initial operating capability of the portals went live the first week of December 2001. Bowstreet's Business Web Factory runs on Sun Microsystems' industry-leading enterprise servers and relies on the iPlanet Directory Server for critical customer profile information.

The portals, at www.performance-coatingsdupont.com, were built to provide auto-body shops with instant access to:

- Marketing tools and promotions
- Online training registration and schedules
- Product and safety information
- Technical manuals and regulatory compliance charts
- Business development tools, including industry performance benchmarking;
- Classified ads and employment services, and
- Industry and customizable personal interest information, such as industry links, news, sports, weather, and stock updates

The portals also provide paint distributors with online ordering and marketing materials to help them solidify relationships with their customers.

Benefits of a Well-Planned Portal

DPC, like many businesses in today's leaner economy, is sensitive to the costs of a major infrastructure investment – especially in the context of what financial managers can view as somewhat fuzzy paybacks. However, the portal is already providing DPC with significant time and cost savings. Online paint ordering is relieving DPC customer service agents from keying phone, fax, and e-mail orders into an enterprise resource planning (ERP) system. Online marketing materials are reducing the cost of printing, managing, distributing, and disposing of paper. Online training registration is saving distributors from clerical work. Online Materials Safety Data Sheets are providing customers with instant access to up-to-the-minute information, while saving DPC from having to fax them to shops. Intangible but even more critical benefits include stronger customer relationships and increased loyalty.

DPC is beginning to leverage the portal globally in line with its original plans. The Bowstreet technology now provides a very customized view for DPC's various brands through a single point of access within North America. DPC's businesses in Europe are using the platform as a starting point for their localized initiatives. Those businesses will use a slightly different view because of their regional business requirements and marketing strategies, but over time DPC expects the platform to fully converge.

DPC recognizes that doing business on the Web is mandatory. Over the next several years it will focus on providing customized tools and industry best practices information to specific customers over the Internet. DuPont wants to understand where its customers' business is strong and where it isn't, and then create value by providing solutions that address those areas. Rather than using the Web as a one-way dispersal of information to customers, DPC is using its portal as a two-way tool for building long-term relationships with its most valuable customers, while simultaneously attracting new clients.

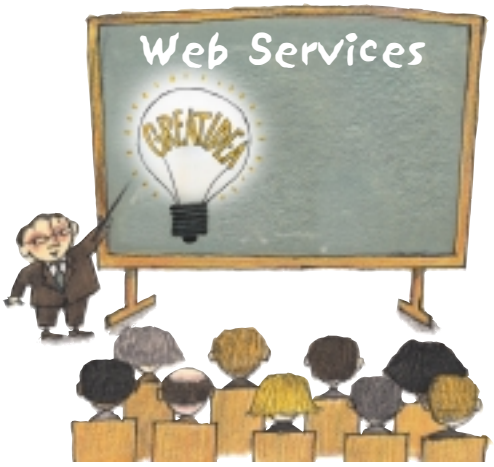
Sonic Software

www.

Web Services Fundamentals: From Whence It Came

A look at the beginning helps reveal future possibilities

Written by Phil Karecki



As the trend towards Web services moves forward, it becomes more important that we understand its roots and origination. Subjects such as Web services, which have a tendency to be technologically discussed, should be viewed from a business angle wherever possible. In particular, the real possibilities of Web services can best be grasped when its foundation and the business needs for it are well understood. With some luck, as my background is technology, I'll be able to shed some light on why this trend has become so powerful and why it now has a chance to succeed in business.

In the Beginning…

The desire to create a layer of technology accessible to all interested and compatible parties has existed for years. Whether you're looking at the operating system to run all other operating systems, standards for communication (EDI is the best example), or the basic concepts behind Java, the goal remains the same – one way for all components to work together. For a variety of reasons, most often self-inflicted, no one standard ever emerged to bring about a common platform for all to use. The goal remained out of reach, also for a number of reasons, but two stand out above the others. First, true cross-platform capability was never achieved. Without a level playing field there was no need to make a change in the IT business model. Second, companies weren't willing to accept the services concept as a means of achieving business functionality. Business was robust, the existing systems worked, and the services concept suffered from the "not invented here" syndrome. Without a commonly-accepted platform and usage by a significant sector of business (any business), there was no way to achieve a common services-based platform.

A significant business driver (the compelling event, if you would) was required to move things towards common ground. That event took shape in the Internet Revolution, and drove both business and technology to where we are today. Description of how and what took place is available elsewhere (on the Net of course), so there's little need to belabor the point. A simple review of popular paradigms (see Table 1) is enough to map the path forward. As the Internet took hold in the business sector, the conversations were all about e-commerce and the ability to sell to the consumer. The term sold the capabilities of the Internet short and we soon moved on to e-business, which was the extension of the technology into business and further into the supply chain. The "adult industry" aside, most of the work and success on the Internet has been accomplished here. More recently we've seen the emergence of *collaborative commerce* and the *networked enterprise* as terms to describe the business and technology trends. Both begin to look at the Internet as a means to make businesses more efficient in their day-to-day processes. Collaborative commerce focuses on two or more organizations working together to achieve those efficiencies, while networked enterprise focuses on making the process more efficient, regardless of where the functions are performed, in essence, this creates a single enterprise view across the partners.

Why Now?

The drivers just described were business drivers, and while they are critical to success we wouldn't be talking about Web services if there weren't some changes in the technologies. For large organizations, the ability to adopt new technology or functionality was often determined by whether the existing platforms would support it. Decisions were based on the hardware investment rather than the functionality offered by a given solution. You wrote or bought for what you owned, and the platform drove the direction.

Think about the challenge that presents itself when the goal is to create a single platform for anyone to use. Until the technology had matured into a better base of standards, a common ground would be impossible to achieve. Well, technology has changed.

The availability (and most importantly, the growing acceptance) of common, platform-independent, tools and technologies was required to support true Networked Enterprises and Web services. With Java and XML (the two primary standards for purposes of Web services growth, Microsoft not withstanding) emerging as truly platform-independent technologies, we were primed to extend applications to areas never before reached. Think of it: now we could write applications purely for the functionality they'd deliver. No longer tied to a platform and the quirks required to code for it, a solution could be written to meet specific industry needs. In theory, hardware could be swapped out under the application with no change to the code itself.

Author Bio

Phil Karecki is a partner with CSC Consulting Group and is the chief architect for Collaborative Commerce. He is a frequent lecturer on cutting-edge technologies, and has been responsible for the development of many B2B sites over the past few years.
PKARECKI@CSC.COM

TABLE 1: POPULAR INTERNET PARADIGMS

Net "Paradigm"	Focus	Reach
e-Commerce	B2C	Primarily buying and selling, to consumers, with the Internet as the medium. Gained a lot of press and drove much of the Wall Street frenzy around the Internet. Much has changed and a number of highly visible names are no longer in business, as they were not able to evolve with the industry or had failed to create a true enterprise-wide model.
e-Business	B2B	Extend the e-Commerce model to handle business-to-business transactions. Buying and selling of goods with the goal of extending the transaction as far into the supply chain as possible.
Collaborative Commerce	B2B	The evolution of e-Business and recognition that success was ultimately achieved beyond the transaction itself. Value had to be delivered beyond buying and selling. Provided a view into the supply chain for partners in business. The process, not the transaction, became the focus.
Networked Enterprise	E2E	Viewing the business beyond the four walls of individual companies and across the Enterprise(s) to deliver value. No longer were solutions ensconced in one place. Now the functionality needed was performed where it was delivered best. This evolution has created the Web services opportunity.

Performance, scalability, and reliability could be dealt with at the hardware level and increased as the business demanded it, without negative or intrusive application impact. Achieving platform independence for the applications opened up a host of other opportunities for the developer and for the businesses themselves.

Object-oriented development, well established (if not widely standardized) by now, had created concepts of modular design and development. Java and XML, with other technologies (the escalating availability of bandwidth, the adoption of the Internet as an accepted part of business, and the slow emergence of real standards included), had provided the ability to create platform-independent applications. And users had become accustomed to the concept that it was functionality that counted, and a single interface (the browser in most cases) was the only tool needed to implement it. Then three separate tech breakthroughs – modular design, platform independence, and single common interface – each beneficial on its own, combined to create a world of possibilities (see Table 2).

Where Does That Leave Us?

Web services have become a viable paradigm due to the emergence of technology and the increasing willingness of business to consider new alternatives for development. With applications no longer tied to the platform, they can truly be developed once and modified only when new functionality or business needs dictate. Updates to operating systems, changes in platform, or new business acquisitions no longer need to force a change in applications. Technology (XML, J2EE, SOAP, UDDI, WSUI, BPML, and other emerging standards) has focused on platform independence and emerged with the ability to deliver that freedom. Combine that with the "need for speed" demonstrated by the Internet, and a business climate that's focused on efficiency and reduced cost, and the interest in Web services becomes clear.

Web services will offer a greater amount of choice (true best-of-breed potential at a reduced cost) to business, and with limited risk. Try it, and if it doesn't do as you expect, disconnect from that service – whether internally or externally provided. Decisions will be based increasingly on business needs and less on existing environment, as companies compete to become efficient within their enterprise and in collaboration with their partners. Web services puts a name to a long-held vision, and the technology finally exists to deliver that vision.

TABLE 2: TECHNOLOGY BREAKTHROUGHS

Technology Break	Focus	Benefit
Modular Design	Code created that can be reused, unplugged, and changed with minimal system impact.	The ability to create programs that can take advatage of the best code and functionality wherever it exists. Sure, initially this would speed development times and encourage reuse, but it set the stage for Web service adoption.
Platform Independence	Develop once Deploy anywhere	Nirvana for the tech. Applications could be written once and deployed on any platform that supported the standard it was written to. True, we have yet to achieve 100% independence, but the trend is big enough to be real and achievable in the foreseeable future.
Single Common Interface	Transparency for users and systems to the location of the application.	Users, or applications, only need to know what function / application / data source they need in order to use it. They don't need additional logins, translations, or knowledge of the system and its location.
Combination	Flexible design runs anywhere single entry points	Frees the business to choose functionality (Web services) from the best sources. Regardless of platform or location. Functionality, whether business-or-technology-driven, that needs to remain in house can do so, while other components can be outsourced as needed without significant intrusion on existing applications.

SilverStream

www.

WSJ FEATURE

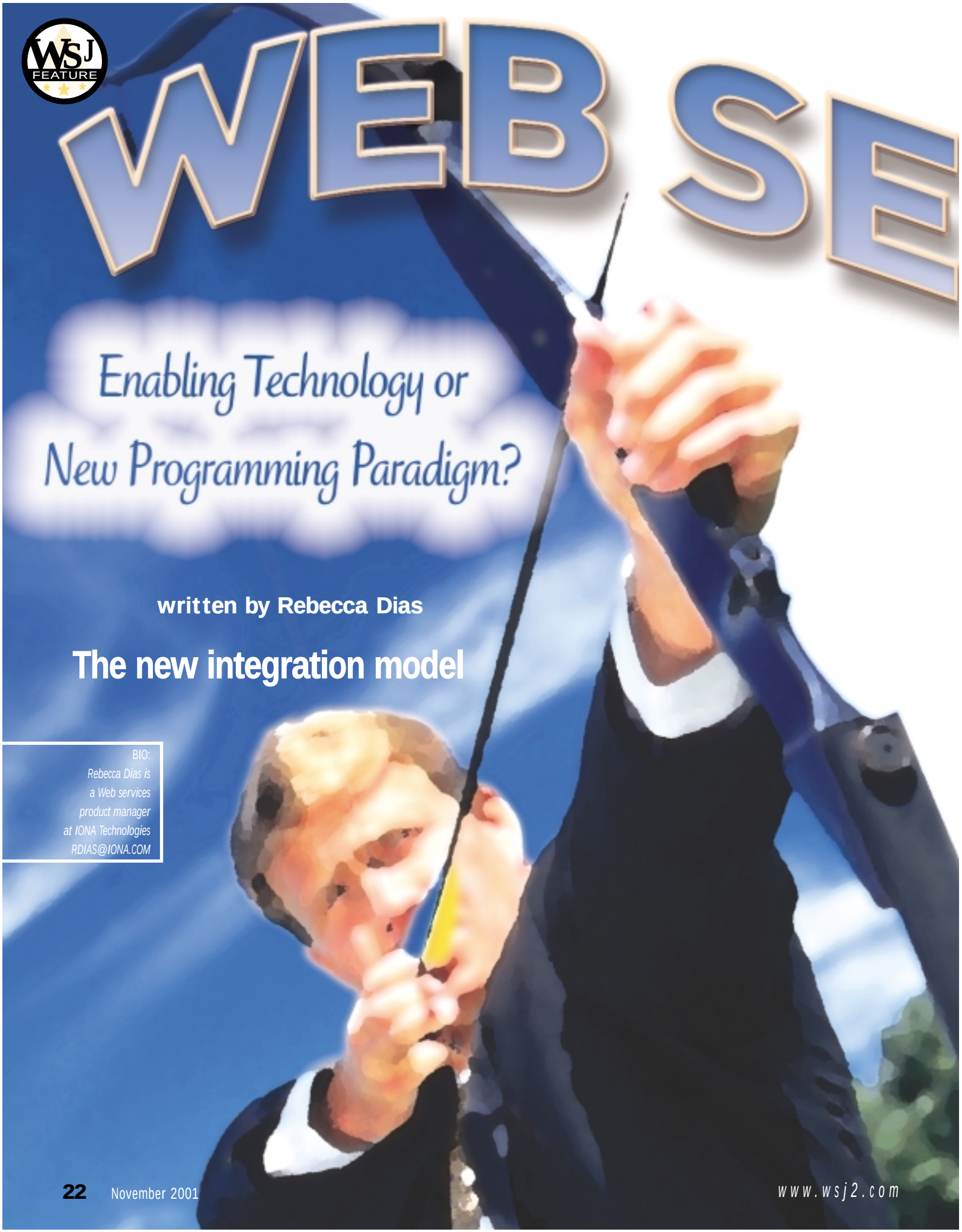
WEB SE

Enabling Technology or New Programming Paradigm?

written by Rebecca Dias

The new integration model

BIO:
Rebecca Dias is a Web services product manager at IONA Technologies
RDIAS@IONA.COM



Starting with a brief look at how developers build systems in J2EE and CORBA, I'll compare these two well-understood approaches to Web Services. Then I'll give a real-world example of Web services. By fleshing out this example, we'll see the business and technical issues relative to building systems based on Web services, and identify the best model for building Web services to solve real business problems.

RVICES:

Distributed Computing Paradigms

When building EJBs in a J2EE environment, developers focus on server-side business components. Decisions are made about the individual components: What type of data is being passed. What is the messaging model, document exchange or RPCs? Is the component stateful or stateless? Is it long-lived or short-lived? Does it need to access the database? The answer to these questions results in the creation of a message, entity, or session bean. If it's stateful, decisions are made on how the state is made persistent. The approach is to identify what type of bean you want to build and then implement the corresponding home and remote interfaces. The exciting thing about J2EE is that after you answer these questions, the primary focus is on building the implementation logic. The container takes care of abstracting away all of the qualities of service such as transactions, persistence, and threading. After the bean is created, it's deployed and can be found by searching through JNDI.

With CORBA, the focus starts with the client contract, the IDL. The contract defines the services being provided from the client's perspective. As long as you implement the contract, the actual underlying details are up to the developer. The server developer leverages prebuilt CORBA services to tackle implementation problems. The designer also specifies the corresponding POA policies used by the server for managing CORBA servants and obtaining higher levels of scalability and performance. Further decisions may be made about threading. After the server is implemented it's registered to a location daemon, naming service, or trader service.

With Web services, you begin by thinking about process collaboration. How will systems integrate this Web service? Based on the process collaboration requirements, how can existing logic be exposed to satisfy this new avenue for accomplishing the business function? The business objective leads the developer to think primarily about integration. Whereas with EJBs and CORBA components the focus is on business logic, with Web services the focus is on the process collaboration. Web services can be found by looking through UDDI or ebXML registries, but like CORBA, the registry isn't necessary because the location of the Web service may already be well known. Let's look at a real-world example to see this model in use.

Real-World Example

The Business Problem

A hypothetical application is presented in Figure 1 depicting a Web service collaboration between a large telecom company with a cellular network called Big Telco (BT) and a music label called Enormous Music Label (EML). To beef up the image of BT's cell phone service and differentiate themselves from the competition, BT wants to enable their customers to not only listen to MP3s on their cell phones, but to allow their customers to browse for and select MP3s to be downloaded to their phone or to a friend's phone. Ordering MP3s for the cell phone must be inexpensive and convenient for consumers or they won't bite. In order to justify the ROI for building such a system, it must be fully automated, easily built, easy to maintain, and have a scalable business model, one that easily accommodates changes in business parameters.

EML wouldn't profit by selling MP3s directly to the individual because of the billing issues.

If an MP3 can be purchased for a dollar, it's economically unfeasible for EML to either take a credit card or send a "snail mail" bill. EML is more than happy to let a third party solve the billing issue. BT already sends a direct monthly cell phone bill to the customer. Customers pay on average $50–150 per bill. The joy of being able to get any song, any time, at a negligible price would ensure that each customer would purchase songs without caring about the additional charges on their bill.

EML isn't in the business of cell phone portals and telecom companies don't have the ability to deal with music copyrights. For BT to provide this service to their customers, they need a partnership with EML.

The Collaboration

In Figure 1, automated contract negotiation occurs at the beginning of the collaboration process. Contract negotiation is an important part of many business collaborations. We've always dreamed that contract negotiation could be automated, but as most people know, it will take years to get to this point. People

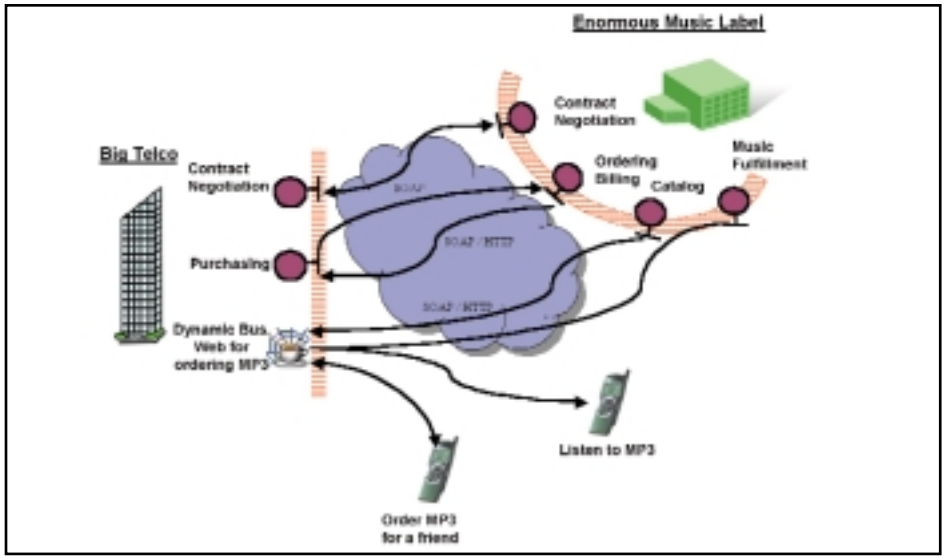


FIGURE 1 | **Collaboration model**

are still needed to do the negotiating to incorporate the proper business and legal processes that ensure a binding and realistic contract. So let's ignore contract negotiation for the first go-round and consider it a future possibility.

The use case is a customer browsing the music catalog on their cell phone on BT's portal to order a song for a friend. The catalog originates from EML's site. BT already has the cellular portal in place to properly paginate Web pages to the cell phone and can easily stream the catalog to the phone once received from EML. BT simply needs to retrieve the catalog information and display it on their portal.

When the customer selects a song to order, BT notifies EML. This transaction needs to be secured. EML has an account for BT and bills them quarterly. The billing system notifies the fulfillment service and sends the corresponding MP3 to Telco, which in turn sends the MP3 to the friend's cell phone.

Either the friend's cell phone receives the song and sends a receipt or the order times out. This is a complicated transaction requiring a two-phase commit across two organizations. If the friend is on vacation without a charger and doesn't turn the phone on for five days and never receives the song, EML shouldn't debit BT's account, nor should BT debit the customer's bill.

Let's look at the touch points of integration:

1. EML cataloging system with BT portal
2. BT portal and customer billing system
3. BT portal and purchasing system
4. BT purchasing and EML ordering systems
5. EML ordering and billing systems
6. EML ordering and fulfillment systems
7. EML and BT fulfillment systems

The reality is that most of the business logic needed to create this system has been in place for years. EML already has catalog, ordering, and billing systems. The system to deliver MP3s to BT is a new Web service with new logic. BT already has the Web portal and ordering, billing, purchasing, and payment systems. The primary additions are the new interface into the back end, the collaboration process, the additional security, the audit trail, the shared transaction, and the pipe.

It's easy to imagine that EML has Macintosh and Windows boxes hosting applications built in Java and Visual Basic. They have packaged SAP applications and a mainframe where the catalogs are archived. It is also easy to imagine that BT is a CORBA shop that recently adopted J2EE for servicing their Web portal. The semantics of these distinct technology platforms need to be abstracted away. Data needs to flow between all of these systems in order for the application to function.

The complexity of the system is apparent when you realize that creating these Web services requires the exposure of many back-end systems that weren't previously accessible via the Internet. Exposing these systems leads to a major security challenge. Both companies have a back-end integration challenge to expose these systems to their new partner. Once these systems are exposed as Web services, the B2B collaboration needs to be implemented. Since BT has the relationship with the customer, they'll drive the overall collaboration. If the MP3 doesn't reach the friend, the customer will look to BT to find out why and will expect not to be billed.

Handcoding this solution would become too costly and prove to be a lost investment. It would be a considerable effort to hand code each integration point in the original design. Once the system is deployed, considerable investment would be required to maintain, upgrade, and customize the system. Eventually BT will want to provide more third party services for their cell phone users and may need to customize the way the Web services interface into their back-end systems to integrate with the new partner. If the relationship breaks down between EML and BT because EML is unable to meet the contracted service level agreements, BT may decide to switch to a completely new partner. Each one of these changes will cause considerable changes in the integration. Hand-coding this integration is possible, but it would be time-consuming and wouldn't scale with the business model.

Tools are needed to aggregate back-end resources to the Web as Web services. Tools are also needed to define the collaboration between the two companies. The corresponding runtime environment is needed to monitor the business process collaboration and ensure that transactions are properly committed or rolled back.

EML needs to ensure that there is ROI in exposing their systems as Web services. Like BT, they'll require tools to integrate, secure, and expose their back-end systems for consumption. Most likely, BT won't be the only customer of this system. Many third parties will also distribute MP3s. These third parties would likely aggregate catalogs from multiple music labels and deliver them to cell phones, cable boxes, handheld devices, and desktop computers. The number of integration points quickly increases and becomes extremely complex, effectively creating a web of Web services. Taking a programmatic approach to accomplishing this integration wouldn't be feasible and would quickly become unmanageable.

The Model

Building a system based on Web services requires that you take the following steps:

1. Expose the back-end systems as Web services.
2. Define the business collaboration model.

Exposing back-end systems is a non-trivial process. Web services are high-level abstractions of back-end systems hiding the semantics of the underlying infrastructure. They need to have simple, easily understood interfaces so when companies begin to consume them, there are no discrepancies regarding the functionality they offer to minimize the integration challenges. The business logic already exists; the issue is how to interface into the existing logic, exposing it as a service. This may require extensive interface refactoring, enterprise application integration, and business process flows.

Interface refactoring can be simple and complex. Only a subset of the overall functionality of the back-end system will be exposed. For example, EML will allow BT to get a copy of the catalog, but not modify it. Data types may need extensive restructuring. Imagine an EJB that passes Java Serialized Objects. It makes little sense to allow a Java Serialized Object to traverse the Web service because a consumer that is built on a VB application or a .NET platform will not be able to consume it.

A single order may have multiple back-end touch points requiring a process engine to tie the systems together and expose the process as a single coherent service. Different systems expect data to be formatted in different ways, transformation technologies may be needed to change the structure of a document for use by different systems.

Sun initiated a number of Java Community Process JSRs, commonly referred to as the JAX APIs. These JSRs define APIs for portable coding to Web services standards. JAX allows for programmatically creating and accessing Web services in Java but in reality much of this is already automated. The reality is that most Web services today are not being built from scratch. As seen in the example, over 90% of the business logic existed before. EAI systems have taught us that it's better to leverage tools to achieve integration than to hand-code it. Hand-coding is less flexible when changes are required.

Once these Web services are exposed, the collaboration needs to be defined. This is also a nontrivial process. If Web services previously existed and were built for another customer, translation may be needed. Often a Web service interface exposed for one customer will need refactoring or translation for another customer to integrate it. Transactional and security contexts may need to be exchanged by these two vendors. If one vendor is using the Microsoft Transaction Service and the other is using the Object Transaction Service, there needs to be a common standard for passing the transaction context to properly interoperate with these systems. SSL can be used to authenticate and encrypt the messages, but if higher levels of security are needed, a standard like S2ML may be adopted to pass security context from one system to the another.

Conclusion

Web services represent a new variant of distributed computing where integration is the main focus. Through our real-world example, you can easily see that Web services present an integration challenge requiring much more than simply enabling existing applications to speak SOAP. The reality is that Web services represent a new Integration Model. Web services require process collaboration, security, transactions, translation, monitoring, logging, and connections into multiple back-end systems. The primary focus of Web services is to repurpose an existing system to enable a company to add a new line of revenue. To justify the investment in these systems, the proper tools must be provided to expose existing systems easily and ensure that integration points can be tied together quickly, translating from one interface to another and transforming one data format to another in a secure and potentially transactional manner.

Written by David Russell

Understanding the Depth and Breadth of the Business Opportunity

Revenue growth and improved efficiency for all involved

A quick dip into your favorite English dictionary will tell you that the word collaborate derives from the Latin *cum laborare*, which simply translates to "labor together." While this is a fine sentiment, in a business context one must consider more than the simple etymology of the words. Had I occasion to pen the dictionary definition of electronic business collaboration I might simply state: "Electronic Business Collaboration: a business model driving a broad spectrum of efficiency gains predicated on the support of open, distributed technology infrastructure."

Many of the articles in this journal will discuss the "hows" of Web services technology, the leading offering in open, distributed Internet technology. This article will consider the possibilities afforded by this technological evolution.

The pervasive spirit behind initial Internet-based electronic commerce efforts was to drive suppliers to lower their prices principally by pitting them against one another in what approximated lowest-bidder auction scenarios in pure trading marketplaces. This model suffers from a number of fundamental limitations. The most obvious is the fact that driving prices downward has inherent boundaries beyond which no supplier can sustain a viable business model.

A model that yields greater efficiency gains driving both revenue growth and reduced costs across a wide cross-section of business activities promises to deliver a much more sustainable and valuable proposition. When all participants across the "value chain" share in such efficiencies, the impact becomes fundamental across entire industries.

The Depth of the Business Opportunity

Electronic business collaboration drives the efficiencies associated with electronic automation and streamlined information management across a broad spectrum of common business activities. Companies have long engaged in diverse cooperative activities, from jointly developed products and information sharing to price negotiations and shared projected demand forecasts, which make up some of the essential elements of successful inter-business interaction. Effective collaboration, therefore, has a fundamental dependency on process alignment and information sharing across partner systems.

The electronic business collaboration model provides value in two basic ways:

- Drives revenue growth
- Improves cost efficiencies

Given such lofty claims, it is best we consider each in turn.

Drives Revenue Growth

Revenues frequently are highly sensitive to time constraints. The ability to effectively execute and deliver on planned business activities can have significant impact on projected and realized revenue streams. Through better management of the overall process comprising a complete set of related business activities across partners, electronic collaboration improves the overall business lifecycle and time-to-market.

A key component of effective process management is the capacity for managing change, adaptive process management if you will. Electronic collaboration makes the task of identifying bottlenecks, modelling alternatives, seizing new opportunities, and implementing rapid effective change across even complex systems a possibility.

Considering the broad array of cooperative business activities that can be impacted positively through electronic collaboration, even incremental gains across each category can result in impressive overall returns.

Improves Cost Efficiencies

A readily recognizable competitive advantage across industries is the ability of any company to effectively manage its costs and overheads. A streamlined operation that is capable of minimizing buffer or backlog either in goods/inventory or better customer and partner interactions is likely to realize a more cost-effective business structure.

Collaborative technologies deliver increased availability of critical business information to the parties that most need that information. Timely information can be applied to aid the reduction of costly buffers and the prevention of costly backlog, or simply to remain one step ahead of potential problems or inefficiencies in dealing with customers and partners.

Again, even incremental gains compound value across the broad spectrum of business activities that combine to effect full business systems.

It is the combined effect of both driving revenue growth and operational efficiency gains that make the electronic collaboration model such a strong value proposition for those that choose to engage it.

The Breadth of the Business Opportunity

So far this discussion has generalized the application of electronic collaboration across business activities without considering these act-

Author Bio

David Russell is CTO and cofounder of Bind Systems (www.bindsys.com), a software vendor developing Web services technologies to enable collaborative electronic business. Bind Systems develops the BindPartner Business Collaboration Platform. David lives and works in Dublin, Ireland. DR@BINDSYS.COM

ivities in any detail. Here I will bring into clearer focus concrete examples of common business activities and consider the value proposition of the electronic collaboration model.

Consider collaborative product prototyping, design, and development. Effective management of the entire process leading from prototyping to development generates a competitive advantage by creating products in less time, at less cost, and with fewer defects.

Given the nature of this type of endeavor, a few key areas of cooperation can be immediately spelled out: the need for shared design and production information, and the requirement that change, both incremental and fundamental, can be accommodated through simulation and/or active process redesign. The electronic collaboration model considers and aids these very types of demands.

The same benefits can be realized throughout a broad spectrum of outward-facing applications employed by business systems. Consider the advantages of the timely and targeted availability of pertinent information considerate of all parties involved in collaborative planning, forecasting, and replenishment activities. Effective international trade and transport logistics can be greatly facilitated through high information availability and adaptive process management.

As a final example, let's consider the demands that arise following a business merger or acquisition. A critical task likely to follow is the alignment of business activities between the two business entities. The ability to not just colocate but also to seamlessly align these activities is essential both where cost is concerned and in the requirement to present a unified front for the resultant business entity. With no guarantees that business process or IT infrastructure will be compatible, a flexible model is required to permit rapid alignment. Again, collaboration rather than tight integration delivers the most expedient option.

Conclusion

Collaboration has and always will exist between businesses regardless of industry. Electronic business collaboration describes a model where such collaborations are facilitated through an evolution in technology and infrastructure permitting inter-enterprise process automation, adaptive process management, and rapid data and information sharing. Such IT advances when applied to business can help drive revenue growth and improve operating efficiencies for all parties involved.

The Web services computing model is delivering the framework for the technology infrastructure and tools required to facilitate the essential loose technology coupling between businesses necessary for open, dynamic yet secure electronic collaborations. An article I wrote for the preview issue of ***Web Services Journal***, entitled "Electronic Business XML: Making Web Services Work for Business" provides concrete discussion detailing exactly how Web services computing technologies are being applied to address the types of business needs considered here.

Mission Critical Web Services

Mission impossible?

Written by Norbert Mikula

Web services are a great vision to talk about. as evidenced by the increasing number of companies declaring themselves the leader in the Web services market. Hype aside, just as with XML, sooner or later we'll all realize there's no Web services market per se but only ways to apply Web services as part of B2B machine-to-machine integration, enterprise portals, knowledge management, marketplaces, self-service forms, and so on. In other words, what we need to focus on is how to use Web services to solve specific technical problems rather than getting excited about new and more dynamic "plumbing."

Returning to the vision part, it's easy to be almost overwhelmed by the many technologies that need to be in place to make the full vision of Web services a reality. The good news is that not all of these specifications and standards need to be in place to get started. But first, let's look at the technology landscape today and how it affects the delivery of the vision of Web services.

Technology Landscape

Interface Standards/Vertical Business Objects

As we know, SOAP can be deployed following a more messaging-oriented paradigm or a more remote procedure call (RPC) paradigm. There are others also, but let's focus on these two.

If we use SOAP for RPC, we don't need vertical standards per se as we're *just* calling methods on remote objects. What worries me a bit is that we see many wizards and tutorials that can easily transform Java objects or COM objects to a Web service, which in itself leads to RPC-type Web services to execute a particular transaction. Because we're calling a method and its associated parameters, I still consider RPC-type calls as tighter coupling.

This is great news if I just want to integrate the Web services of one or two business partners. It becomes more problematic if what we're interested in is transparent information exchange across an entire value chain, where we need to have agreements on messaging structures rather than just APIs.

On the side of vertical business objects, thanks to the work of the numerous industry consortia, we already have a pretty good set of business objects available and we're starting to see the adoption of some of these.

Web Services Security and SSO

What's the most annoying thing in dealing with Internet e-business services? Did you think "having to log into each system (over and over again)"? Right!

The magic word is *single-sign-on*. It's a core part of the Web services vision to be able to integrate and aggregate services provided by numerous geographically and logically dispersed providers.

Without standards to provide means for services to exchange users' authentication information, large-scale deployment of dynamically aggregated services won't happen.

The good news is that we have a few very promising initiatives going on. One is the OASIS Security Assertions Markup Language; the other is Passport, driven by Microsoft.

Distributed Transactions

Gone are the days when ACID was everything we had to worry about (and I don't mean kids doing drugs in discos). Atomicity, Consistency, Isolation, Durability were everything we needed to ensure well behaved transactions in our systems.

It became more complex as we started to distribute our systems. All distributed databases and other such systems today support "two-phase commits" to deal with the increased complexity of transactions in distributed environments. In this scenario, all participating processing nodes have to either commit or rollback a particular transaction.

The Internet, however, is the most distributed and complex system we can imagine. Two-phase commits alone are not suitable to address the complexity of Internet-scale transactions. In the Internet realm we typically deal with multiple entities (customers, suppliers, and business partners) that use loosely coupled Web services to execute a particular transaction. This overall transaction in turn consists of smaller transactions that are wired by some implicitly or explicitly defined workflow. Should an individual transaction fail, we may not be able to roll it back so we have to introduce a compensating transaction to regain system integrity.

While the standards and vendor community is working hard to address this problem, no broadly accepted standards exist. Examples of interesting efforts in this arena include the OASIS Business Transaction Protocol.

Business Processes

"Workflow goes Internet." While in the past we could be content with describing the workflow within the confines of our corporate boundaries, today we have to deal with complex interactions between a number of parties involved in our extended enterprise (including customers, suppliers, and business partners).

Emerging specifications will help us find a common, reusable, and hopefully interchangeable format for describing these processes. The most promising initiatives include the OASIS/UN-CEFACT ebXML suite, Rosetta Net, Biztalk, BPMI, and WSFL, just to name a few.

In this category of emerging technologies, the difficulty isn't finding novel ways of addressing

Author Bio

Norbert Mikula is a founding member of DataChannel and is 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 and speaks regularly at industry events.
NORBERT@DATACHANNEL.COM

the problem, but rather coming up with a broadly agreed upon way of doing it.

On a smaller scale, we'll need business process management as well. Web services help us to break up applications into discrete components. Once that has happened, something has to integrate them back together. This will be either some glue-code or – even better – some externalized business rules driven by the business process engines.

Reliable Protocols

We want our systems and our Web services to be reliable. The problem is, while we know that SOAP doesn't care about the transport protocol used to get a SOAP message from A to B, we also know that we'll use HTTP, especially between companies.

Unfortunately, while HTTP is known for being ubiquitous, it isn't known for being reliable. Two approaches can be used to fix this. You can build code around these protocols – which is what most designers have to do today – or you can make the protocol more reliable, which is what the guys at IBM are suggesting with their "Reliable HTTP" – httpr.

Payment Services

Software as a service? This is unlikely to work without an associated plan for charging for it. (Larger software companies seem to prefer this model to the old-fashioned buy-and-own.) Certainly a recurring revenue-stream seems to be valuable, and it will be much harder to do a "backup" copy for a service-based software package.

As we talk about payment services, however, we need to distinguish carefully between paying for software and the use of it (as described above), or whether we're talking about a business service (like a credit card check) that we pay for (whereas the Web service is more the means than anything else). I'm not yet convinced we can treat them in the same way (but I'm always willing to listen).

Web Services Adoption

So are we doomed? Will we have to wait for all these things to be defined and standardized? Not really!

Informational Web Services

These kinds of services are the best examples for using Web services without having anything else in place. Not mission critical in nature, these services offer stock quotes, weather reports, traffic reports, and so on. In this case I am not really concerned with anything else but getting to information (and showing off "the power of dynamic Web services").

These services lend themselves nicely to RPC-type SOAP calls where we want instant gratification. I believe all the services on the well-known Xmethods site follow this paradigm without exception (which isn't surprising).

Web Services as Next Generation APIs

This should be the most straightforward thing to do (especially for applications that aren't tied to a heavy user interface component). Is there an existing SDK to your product? If yes, then all you need to do is wrap the EJP, JSP, COM, or whatever components you use into a collection of Web services and suddenly your product is Web services–enabled and Internet-ready.

Extending the Reach

This is where it gets really exciting with respect to Web services in the short term. In the past I could have a few online forms as part of my extranet and people would use them to order goods, check on availability or delivery, and so forth.

What limited the reach of these applications was that, for one, all of the user interface was hardwired. Many intranets have set guidelines for colors, fonts, and other style-related attributes. With Web services you can integrate a service and have complete control over the visual appearance, providing seamless integration.

Secondly, interfaces weren't explicitly described. This made it hard to programmatically call an old-style Web service without having to dig into the HTML code to figure out what the parameters were (not talk about valid types, error-handling, etc.).

With such Web services, it's also easier to extend the reach beyond the browser to cell phones, PDAs, and all the other great devices that are still looking for the "killer application."

Small to Medium Enterprises

With Web services, a smaller enterprise can now do what it previously would have required costly EDI infrastructure for. Now it can finally afford electronic data interchange and participation in larger markets (we just need to wait for the right tools to hit the market). The OASIS/UN-CEFACT was designed with this category of users in mind and it's also architected to use Web services from a transport perspective.

Dynamic Trading Networks

In the short term we'll see some vendors providing platforms to develop, publish, locate, integrate, and charge for Web services. Since none of the standards I described are in place, these vendors will have to make up for some shortcomings by "proprietary" approaches.

This is a part of the vision of dynamic Web services where we see most disconnect between current practice and future promise. The problem with this vision is that, while exciting and promising, it does have to overcome many existing hurdles of today's business and technical everyday practices.

A company's data structures (read *schemas*) contain a lot of information about how they do business (often, information they want to share with just a handful of business partners). The same holds true for business processes or other aspects of a company.

Maybe that's why the adoption rate of public service and schema registries is so slow (and I am not talking about quantities of objects stored, but rather the way people use them and for what purpose). Unless you're *really* interested in the vision of dynamic discovery of business services, you're more likely to just publish your Web service descriptions, schemas, and other interchange relevant components to your known business partners, and that's all you need to do..

Trust

Trust has been at the heart of business since the dawn of time. The "new economy" requires a company to trust an unknown, almost anonymous entity to deliver goods or (Web) services that are critical for its business. That's a lot to ask for, which is probably one of the reasons why, according to the various assessments, private and more controlled e-marketplaces are more pervasive than open and "public" ones.

Summary

We can see Web services beginning to emerge as a means to solve business problems. Once we've covered distributed transactions, single sign-on, business process management, and so forth, we can talk about the new world order. Until then, and even then, Web services – just like any other technology – will change the way we do business, but often not as fast as we *i*-technology visionaries would like.

Reviewed by Brian Barbash

About the Author:
Brian R. Barbash is a consultant for the Consulting Group of Computer Sciences Corporation. He specializes in application architecture and development, business and technical analysis, and Web design. RBASH@CSC.COM

CapeConnect Two for J2EE

by Cape Clear

Effectively extending enterprise components

Two distinct initiatives have come to the forefront as means by which applications can be presented as Web-based services: Microsoft's .NET and the J2EE platform. Application servers and toolsets are beginning to tailor themselves to the Web services paradigm as the concept matures. One such product is Cape Clear's CapeConnect Two for J2EE.

CapeConnect provides an environment in which Java and J2EE components may be presented as Web services accessible via SOAP calls, defined by WSDL, and cataloged by UDDI. Figure 1 illustrates CapeConnect's architecture. SOAP clients access the CapeConnect Web Services Platform, which routes requests to the appropriate enterprise component. This provides the capability to leverage the existing investment in enterprise technologies with minimal modifications to those systems.

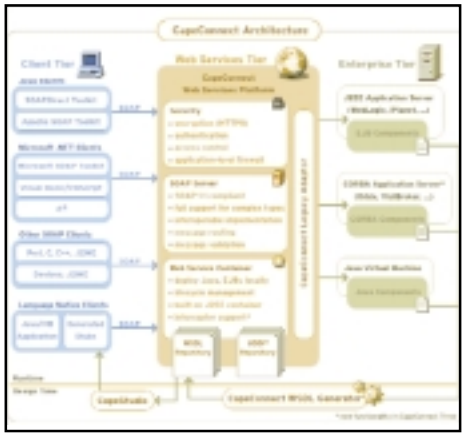


FIGURE 1 | CapeConnect architecture

Components of CapeConnect

CapeConnect consists of three major components:

- The ***CapeConnect Gateway***, essentially the message broker, receives all SOAP client requests and forwards the messages to the XML engine.
- The ***XML engine*** converts the incoming SOAP requests into Java method calls, routes them to the appropriate enterprise components, and converts the results back into SOAP messages. Exceptions are handled through SOAP faults.
- The ***J2EE engine*** is a fully-compliant Enterprise Java container that may be used to serve EJBs. Alternatively, CapeConnect fully integrates with BEA's WebLogic 5.1 and 6.0 application servers, which may be used in place of the provided container. Future versions of CapeConnect will include integration with the iPlanet family of application servers and IBM's WebSphere.

Working With CapeConnect

Installation and Configuration

The installation process for CapeConnect is simple. I've installed it on a Windows 2000 Pentium III machine with 256Mb of RAM; versions are also available for Linux and Solaris. At install time, the choice between using the provided J2EE engine and integrating with WebLogic for serving EJBs is presented. Integrating with WebLogic requires a few simple steps to establish communication between the servers prior to creating services. Once installed, CapeConnect is ready for use with example applications and for deploying custom components. For this review, I've elected to use the WebLogic integration option.

Supporting Web Services

CapeConnect Two for J2EE exposes stateless session EJBs and standard Java classes as services available through SOAP calls. To do so, the developer deploys Java classes and Enterprise beans with CapeConnect either in lieu of a separate application server if the provided container is being used, or, if integrated with WebLogic, as a supplement to the normal deployment process. CapeConnect then generates the stubs and XML translations required to handle requests. Optionally, the developer may generate a WSDL profile for the object being deployed.

Two formats of SOAP action fields are supported by a proprietary format recommended for clients that exclusively access CapeConnect services, or a custom-defined format providing flexibility to clients communicating with multiple server environments. If the custom format is used, CapeConnect requires that WSDL be generated for the components deployed.

Creating Web Services

I've created a simple movie management system using EJBs deployed on WebLogic 6.1. It allows users to create and update a catalog of their personal collections of movies and videos. The system contains a single stateless session EJB to provide the external interface and a BMP entity bean to manage data persistence.

Once the beans are deployed on WebLogic, CapeConnect must be configured to route SOAP requests to and from the stateless session EJB acting as the interface mechanism. The first requirement is the location of the application server that hosts the components. An XML configuration file manages the mappings between services

CAPE CLEAR

CAPE CLEAR
2121 North California Boulevard,
Suite 290
Walnut Creek, CA 94596

$950 per developer
U.S. $10,000 per runtime deployment CPU

FOR SALES INFORMATION:
Tel: (925) 974 3311
E-mail: sales@capeclear.com

and their containers. During installation, if integration with WebLogic was specified the connection to the server is already established in the configuration file. Additional instances may be added by editing the file manually. Once the location of the EJB container is specified, the CapeConnect console may be used to add the individual application object. The information provided in the entry for each service includes the service name, the JNDI name for the associated object in its container, the server on which it resides, and whether the service requires a secure connection from the client. If no WSDL is required, clients may begin requesting the deployed Web service. Otherwise, WSDL may be generated for deployed objects through a wizard.

SOAP Clients with Cape Clear SOAPDirect API

Although it supports custom SOAP messages, CapeConnect provides a proprietary API called SOAPDirect. The API is intended to simplify the development of Web services clients that communicate exclusively with services exposed by CapeConnect. It isolates the details of SOAP messages from developers to reduce the learning curve and time needed to create service clients.

Creating a SOAP Client

Creating SOAP clients with SOAPDirect is very straightforward. The first step in the process is to create a request object and add any associated parameters. The developer is required to have the URI for the desired service available in order to construct the request object. The URI includes the application name specified in the configuration console and the JNDI name of the object being requested. There are three mechanisms to handle parameters for a request.

- ***Simple parameters***, such as Strings, Integers, etc., may be added to the request through overloaded add methods that support the various Java data types.
- ***Array parameters*** may also be passed to the request from additional overloaded methods that receive an array value of the various Java data types.
- ***Generic container classes*** handle more complex, custom structures. This container class holds field names and values for the custom object in the request, which is then mapped to the method signature of the called component.

For the movie example, the lookup method requires two parameters. The first is a primitive integer value that represents the unique number of the movie. This is set with the appropriate add method on the request. The second parameter is a custom user object specific to the movie manager system. In order to pass the user along in the request, an instance of SDComplexType is created. Each property of the user is added to the SDComplexType object, which in turn is appended to the request object.

When the request is invoked, the CapeConnect Gateway receives the request and sends it off to the XML engine where it's interpreted and sent to WebLogic 6.1 for processing. WebLogic's response is received, translated back into a SOAP reply, and sent through the gateway to the calling client as an SDReply object.

From the SDReply object, data values can be extracted via the accessor methods structured not unlike the accessor methods on a JDBC Resultset object. The reply object also provides an accessor to an SDComplexType object that maps any complex object returned from the EJB. In the case of the movie management system, a customized movie object was returned with all the properties of a movie. By providing the name of the desired property, the values may be retrieved and manipulated in subsequent processing.

Summary

CapeConnect Two for J2EE provides an effective solution for extending enterprise components including EJBs, CORBA objects, and standard Java classes to the outside world as Web services. It's a very simple package to install and administer. The provided SOAPDirect API has a reasonable learning curve that isolates developers from the intricacies of SOAP. It also provides the flexibility to work with any variant of SOAP and makes the creation of WSDL definitions extremely simple. With its integration into WebLogic, and future integrations with iPlanet and WebSphere, CapeConnect allows existing investments to be leveraged and extended with minimal impact.

Written by Darren Govoni

Information Syndication Using Web Services

A dynamic way to share data and boost profitability

Current trends in IT and business computing indicate an evolution toward the dynamic exchange of business information, market intelligence, and commercial transactions. Technology standards such as XML have provided the common language needed to interoperate effectively across networks and systems.

In addition, the collection of technologies known as Web services (XML, SOAP, UDDI, WSDL, etc.) provides a complete platform for interbusiness service and data exchange that's rapidly becoming the modus operandi for e-businesses.

The goal of these efforts is twofold yet oddly diametric – on the one hand you have the need to de-couple business systems and infrastructures so they can change freely, and on the other you want to couple disparate networks and systems together on demand. Enterprises use a variety of different technologies to meet their needs and these have often prevented seamless ways of connecting their systems together. To address this, XML-based protocols and schemas have introduced a way to de-couple technical systems, allowing for dynamic exchange of business-specific, self-described data. Yet, the other extent of Web service technologies is to join businesses, not so much technologically but rather in terms of business processes or information.

Popular Web service platforms utilizing UDDI (Universal Description, Discovery, and Integration) permit for the dynamic searching and association of business entities described therein by WSDL (Web Services Markup Language) documents. Specifically, one business system can establish criteria to match entries in a chosen registry, which can lead to spontaneous business transactions with the described entity in an implementation-independent manner.

These capabilities will change the way enterprises interoperate (or enter-operate if you will). One possibility that can emerge from this is the aggregation of business information with the goal of establishing coherent market intelligence among members of a business community. Companies sharing a common interest or objective will find ways to band together and benefit from the ability to contribute to and participate in a common economic model that enhances their market viability as a whole within given vertical sectors. I use the term *information syndication* to refer to the aggregated publishing of business information as a commercial service.

Today, tangible and media commodities can be delivered across a value network via corporate syndication through various channels (see Figure 1). The presence of Web services provides the necessary technology platform for a variety of disparate companies to create a virtual platform with the aim of delivering business information, data, and market intelligence as a combined service. This article explores the idea of information syndication using Web services.

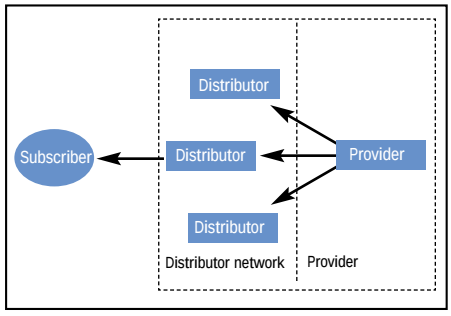


FIGURE 1 **Traditional syndication model**

The Importance of Web Services

Traditionally, enterprise extranets have involved managing vast software, hardware, and human resources. Business extranets are costly to develop, operate, and maintain. Common facilities among extranet partners were often duplicated between their systems, increasing the complexities of the partnership network. Furthermore, security concerns regarding controlled access to information within corporate network boundaries typically meant sensitive (and nonsensitive) information couldn't be shared or transacted since they often existed in the same place. It's difficult to assign segregated policy concerns in a fine-grained way to corporate data, let alone provide external access control to resources within the corporate intranet and continually manage them.

These issues have stymied businesses' ability to come together dynamically in ways that allow for a rich flow of information into and out of corporate networks and repositories. One of the attractions of the Web services platform is the way it hides the details behind the Web services protocols (or interfaces; see Figure 2). This effectively allows companies to publish only the services they intend to offer and to encapsulate acceptable security policies within those services regardless of how a particular user accesses the system (since Web services define this already).

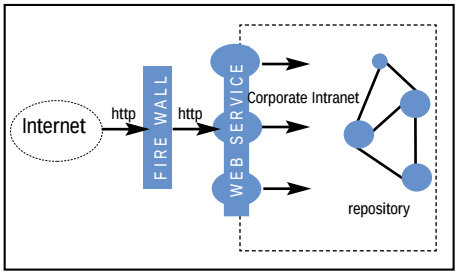


FIGURE 2 **Web service front end with intranets/databases behind**

Because XML is the protocol metalanguage used to describe messages used in Web service conversations, businesses can provide their own semantic meanings to self-described data arriving or retrieved from a variety of sources. This decoupling of data formatting from semantics enables businesses to share common ontologies of data, which can

Author Bio

Darren Govoni is the founder and CTO of Metadapt Design Systems, Inc. Darren currently works for Cacheon, Inc, whose technology is based on technology he developed at Metadapt. DGOVONI@METADAPT.COM

then be turned into information useful for that particular business.

Information Service Providers

In fact, information service companies have been doing this for many years. They've used a variety of means to capture volumes of raw data from various sources, which is then deposited into a database where it can be sorted and formatted for content. Additionally, information subscription services, such as Reuters and AP, online content service providers like the *Wall Street Journal* and Bloomberg, and database service providers such as Lexxus/Nexxus all capitalize on gathering data, converting it to information, and formatting it as content for particular audiences. Often the base data is the same, but enhanced or tailored to particular subscribers.

This is the general model employed by an information syndicate, yet the information is gathered from the syndicate network members and typically distributed via proxy (which we'll discuss later). In traditional media syndication, you often have a single source with multiple distributors. Our information syndication model is the inverse of the traditional approach and therefore I refer to it as the *inverse syndication model* (see Figure 3).

Data Warehousing

Data warehousing technologies attempt to bind disaggregated data into common, semantically coherent sets for search and redistribution. They often do this by normalizing a dictionary of metadata associated with the raw data, which is then converted into searchable, often usable, information. This is

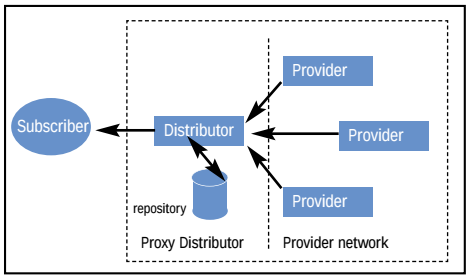


FIGURE 3 | **Inverse Syndication model**

useful for a number of reasons, including qualified searches, more accurate results, and interoperable information.

Different sectors will promote specific metadata most applicable to their data (or industry), and similar companies will seek out common, binding metadata to make their information useful and usable within their community. (Fortunately, this has been going on for a few years now as companies and sectors define rich XML schemas to process intercompany XML-based messages.)

Information service companies, sometimes known as "data marts," will promote these data-unifying efforts and in turn publish their databases to numerous subscribers. In addition to this, information syndicates will seek to promote an economic common interest between the member companies or the market as a whole. In this way, it acts as an avenue to revenue for the data providers who seek to hone their offerings for the advancement of the alliance.

A separate revenue model can be employed on behalf of the syndicate as profits can be redistributed in various ways back to the member providers (in the form of dividends, incentives, and the like). This economic justification is important to lure members into a particular syndicate network, which then, in turn, improves the attractiveness of the system – achieving a natural feedback and scaling effect.

Information Webs

All businesses produce information, yet most don't maintain ways of publishing that information, converting it to valuable content or monetizing it. This stems from some basic economic assessments of the businesses' core competencies. The fact is, many businesses, especially those outside of IT, do not have the capital or expertise to set up complex service provisioning networks, nor the business model to justify housing the technical responsibilities or weathering a separate value network exclusively. Nonetheless, they continue to gather volumes of raw data that can be mined for valued information. The trick is extracting it in a safe and secure manner, and providing an economic justification at the same time.

The Web services platform, including ebXML (electronic business XML), provides the needed pieces to automate service monetization and transactions over the Internet. This provides a scalable, volume-driven commerce model that can balance the costs of establishing the infrastructure. However, it's important to understand the effective demand for the services and information provided by an individual company. From the view of a single company, the economic picture may be difficult to see clearly, yet when the demand is multiplied through the establishment of a strong partnership network, a critical mass can be achieved. This is what information alliances and syndication offer in this context.

Using the Web service approach, companies within (or across) vertical sectors can establish interconnected service networks permitting them to exchange information useful to their markets. However, it may be necessary for members to establish a consortium of interest to carry the group policies and manage the technical aspects of delivering the combined information services on behalf of the individual members. In this case, the need for proxy representation becomes clear.

Proxy Repositories

It will be advantageous for the syndicate alliance to appropriate a proxy service provider to manage the warehouse repository and the policies governing commercialization and distribution of its content. It can do this a number of ways, one of which is to set up and sponsor a separate company (or delegate to one) to handle the duties. By outsourcing this role, the businesses can overcome the economic and technical burdens that would not otherwise mix with their core competencies or revenue models. The proxy service provider might toll transactions against the syndicate and not require any direct service fees or payments. Many interesting business models can be employed here; however, that discussion falls beyond the scope of this particular article.

A proxy service provider will host access to an information syndicate and expose a Web service front-end allowing subscribers to execute complex queries, receive periodic transmissions, and access other useful ways of extracting information from the syndicate members without having to compromise their identity. It'll also provide clusters of repositories containing strategic information obtained from the syndicate members. This externalization of information protects the provider's intranet and databases from anonymous access and shields the technical platform details of the provider's infrastructure from users of the proxy service provider. The use of Web services as an exposure mechanism allows subscribing entities such as individuals or other computing systems to transact and acquire the information automatically.

Publishing Models

There are two primary methods of interfacing with the proxy warehouse as a provider. One involves publishing or proactively pushing information from within your corporate networks directly into the proxy warehouse via a Web service interface (maintained by the proxy service), as shown in Figure 4. This gives you the comfort and

security of deciding what information is targeted and how this actually takes place internally – as well as how often. The other method relies on the proxy warehouse to poll your Web service interface periodically and decide what information to retrieve (see Figure 5). Even then, the individual company can restrict the scope of access from the proxy warehouse. The added benefit of the latter permits the business to simply publish the appropriate compliant Web service front-end that the proxy warehouse will use to contact it.

No additional internal maintenance or management need be done routinely to ensure participation in the network.

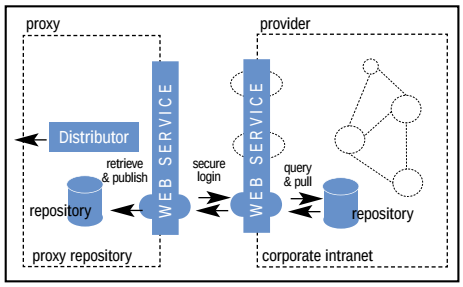


FIGURE 4 | **"Push"-based information publishing model**

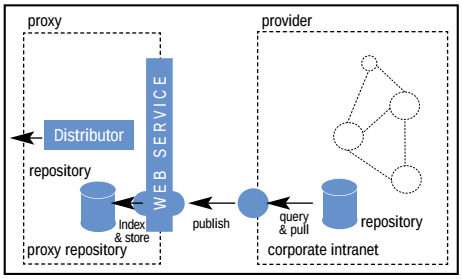


FIGURE 5 | **"Pull"-based information publishing model**

However, this is a more passive approach as you may not be precisely aware in advance of who can be accessing your services, or how access is achieved.

The proxy service provider can also broker requests for information directly into the syndicate members in a dynamic, on-demand basis. In this situation, information seekers access the syndicate network, as a whole, through the proxy service, much as Internet search engines work today. All system interaction occurs via Web service interfaces (or common Web front-ends) and the appropriate access permissions and policies will be in place within and behind the proxy service provider. In fact, its duty is to enforce the concerns and policies of its sponsors.

A Business Scenario

Here's a practical business scenario. There are many different auto service and repair stations throughout any given metropolitan region. If you're looking for a particular tire for your vehicle, depending on which service center you go to, they may or may not have it. Fortunately, most service center computer systems can search other locations and check inventory elsewhere, but only within their own company. Now, suppose a band of service centers run by different owners know they're constantly referring business to one another when supplies are low, yet they reap no reward for this. The owners come together and decide to execute a referral reward system. Whenever some automotive part, in this case a tire, is sold as a referral from an alliance member, a finder's fee is awarded. Now the problem is that they have no way of automating this process since their software systems are all different. They want to form an information syndicate where they can publish to their proxy warehouse only the number of tires they have on hand (maybe they agree on the referral award program only for tires and don't want to expose other inventory details).

Using the approaches we've discussed, each alliance member would publish tire inventory data into a proxy warehouse where other members could subscribe to it and pull it into their software systems. Maybe their policy also allows anyone to subscribe to that information for a fee. The information is updated automatically by each service center's systems using Web services as the glue. Their systems aren't directly connected and therefore are protected from direct access.

Consider the independent outsider, for example, an industry analyst researching the number of tires sold or housed by service centers in a particular region. The analyst too could subscribe to the information through the proxy service provider and get up-to-date and accurate information. Nothing special would be needed as the information could be obtained or delivered in any number of ways, such as common Web content, e-mails, pages, etc. Many other kinds of businesses can benefit from this approach as well. Financial, health care, and market research organizations naturally transact volumes of data to conduct business, but the manner in which they do so doesn't permit for the levels of separation and automation we've presented here, or is more exclusive to the individual parties where greater economic interest might exist.

Conclusion

The dawn of Web service technologies is here, and before sundown companies will be racing to identify how to benefit from the platform and what it has to offer. Ideally, we see an increase in the exchange of data via Web services as it provides an automated way to tap into business information. For many companies, the issue is what to offer and expose, and why. Information syndication can provide a demand-driven market for corporate data gathered and provisioned by large alliance networks. The syndication of information will provide dynamic ways to publish and transact otherwise dormant data for the sake of market intelligence and profitability. It remains to be seen how the critical masses can be achieved and whether the revenue model of information syndication is sustainable beyond certain high-volume verticals. We can expect to find out in the coming years.

WSJ FEATURE

written by Ben Bernhard

Author Bio:

Ben Bernhard is the IONA XMLBus engineering manager. He is responsible for the product's technical directions and implementation. The XMLBus includes a Web Services Builder that allows rapid construction of Web Services from existing systems and the industry's first Web Services Container.

BEN.BERNHARD@IONA.COM

A Web Services

CONTAINER

SEEKING FEATURES FOR THE FUTURE

The Web services paradigm promises enterprises a flexible and robust infrastructure for assembling loosely coupled systems. Organizations can expose existing functionality to trading partners inside the company, within a consortium, and across an industry. Interfaces are well specified using WSDL, and communication is standardized as SOAP messages.

Vendors are rushing to release technology previews showcasing SOAP and WSDL support. These previews are great for learning the new paradigm and implementing toy Web services. For the Web services paradigm to thrive, these technology previews must mature into useful platforms supporting real applications. Press releases already tout the traditional enterprise infrastructure touchstones of performance, scalability, reliability, and security. But many current implementations are limited and don't have the architecture required to support mission-critical systems.

The Web services container provides a first-class environment for deploying, running, and managing Web services. In this article I present the requirements for a Web services container, which addresses the weaknesses of currently available SOAP implementations. The container embodies a collection of features necessary for substantial real-world use of Web services products. Its successful implementation is a prerequisite for products that plan to deliver enterprise Web services features.

Web Service Infrastructure Requirements

Web services will establish new connections between widely disparate computing environments. The industry has done connectivity before and many companies know that enterprise connectivity mandates a certain set of requirements. Experienced vendors will bring to the market Web services products that perform and scale well, and are reliable and secure. But even these traditional product evaluation points have Web service–specific implications:

- ***Performance***: The standard requests-per-second metric is not critical at this stage. Web services hold the promise of automating manual connectivity and replacing e-mail, manually constructed chron jobs, ftp scripts, and even proprietary or custom data exchange over HTTP. The critical resource is not processor time or bandwidth; it's systems development expertise. To be successful, a Web service infrastructure must facilitate radically faster implementation and deployment than current approaches. As the paradigm is adopted and becomes widespread, traditional system throughput metrics will be more relevant. Note that the most applicable measurements won't resemble the test-lab metrics reported by middleware vendors. Rather, look to see quality of service and performance measurements based on real deployments and designed to parallel the numbers collected by application service providers. Sustained

daily transaction volume, median response times, and hourly data flow rates for a deployed system will be better reflections of the real-world performance that can be achieved in a widely distributed environment that includes public networks and geographically distributed systems.

- ***Scalability***: Most current Java Web service implementations are constructed using an existing HTTP server infrastructure. Most are simple servlet code generation systems. Web servers and servlet engines have been carefully tuned over the last few years to provide good scalability across many connections and lots of hits. Unfortunately, these are the wrong metrics. Most Web pages are 10–20k in size. Most HTTP posts come from HTML forms and are 2–4k. The toy Web services published today include mortgage calculators, stock quote services, and mortgage calculators. These demos won't stress scalability. However, when Web services are deployed throughout the enterprise, they'll be carrying messages of 10 or 100 megabytes in both directions. Servlet engines will need to be re-tuned to carry this kind of payload.
- ***Reliability***: Web services sessions are currently stateless. This makes failover and other fault tolerance measures relatively easy to implement. However, sophisticated Web services applications will be constructed and many will require stateful connectivity. Vendors like IONA that have developed clustered, fault-tolerant J2EE, JMS, and CORBA implementations will leverage this infrastruc-ture to deliver the same features to Web services customers. In the near term, look for Web service products to deliver these features by leveraging transports that are more sophisticated than HTTP. JMS, TCP, SMTP, and even FTP come to mind. Enhancements to HTTP, like the proposed HTTPR, will improve the reliability of what is sure to be the most common transport in the short term.
- ***Security***: This one is obvious, but not trivial. It's one thing to let a customer send you a credit card number over the Web. Sending part of your accounts receivable database to a collection service is something else altogether. A security breach that effects one transaction implies radically different liabilities in these two examples. Another crucial point: current business-to-business and enterprise integration solutions connect a relatively small number of well-known participants. As Web services become common, the permutations of participants and service offerings will explode. While authentication will be required, seamless authorization and credentialing system integration will be essential.

The features listed above will be required of deployable Web service implementations; they're required behaviors of any legitimate distributed systems infrastructure. And that's exactly how they fall short as imperatives for Web service infrastructure vendors. These features don't uniquely identify or define Web services. They don't address the key value propositions of this new paradigm. As argued throughout this magazine and elsewhere, the Web services vision is unique and compelling. Implementations will let companies build new systems not previously feasible. The question we must address is: What characteristics are unique to the paradigm and necessary for the success of this new approach to integration?

> "It must be possible to upgrade the Web services platform with minimal impact on running services."

Web Services Characteristics

Most Web services will be constructed to expose existing functionality to new partners. Web services will be gateways with interfaces tailored to specific usage patterns or even specific partners. Where IT systems are updated slowly with new features exposed quarterly, Web services will be constructed using eXtreme Programming techniques and updated biweekly. Web service platforms must support rapid, seamless rollout of new or improved Web services. Furthermore, the most robust platforms will support multiple versions of the same Web service.

In these early days the Web services infrastructure itself will evolve rapidly. IT systems may upgrade their infrastructure yearly, but groups implementing Web services will want to move immediately to the latest release of their chosen Web service platform to gain improved interoperability, new data type support, and enhancements in the enterprise characteristics listed above. It must be possible to upgrade the Web services platform with minimal impact on running services.

Many Web services will be implemented using HTTP as the SOAP transport. However, some early adopters will require (or discover that they require) the high quality of service that can only be delivered in the short term by using a more mature transport.

Standards-based Web services won't be constructed by hand, but rather assembled using an infrastructure product such as IONA's XMLBus. This is possibly the most significant benefit to be gained from WSDL, SOAP, and related JCP initiatives. Proprietary products have simmered for several years now and IT organizations have been manually building a few Internet-based services with some success. For example, the U.S. Postal Service has an offering at www.uspswebtools.com. These early Web services efforts have been relatively few because implementation requires a high level of skill, knowledge of cutting-edge technologies, and often a willingness to commit to a vendor's proprietary solution. The latter characteristic is perhaps the most painful, because efforts not based on standards offer no hope of interoperability.

The maturing Web services standards let vendors construct and promote competing products that are interoperable, comparable, and offer enormous developer productivity gains. They allow IT organizations to benefit from this paradigm without vendor lock-in and without developing the high levels of expertise required to build a custom solution. In short, the standards are central to the creation of a new infrastructure market. Successful projects will employ complete solutions that deliver an easy-to-use infrastructure for developing and deploying Web services.

Web services are new and not widely understood. Most IT organizations have no experience with or knowledge of the related standards. Widespread adoption can't be achieved by small groups of specialized consultants. How is it possible that this new paradigm will become common infrastructure?

The answer is that current IT staff will expose existing systems as Web services. These developers have an intimate understanding of existing systems and in many cases a solid understanding of distributed development or systems integration. This current skill set is largely sufficient. Successful Web services products won't impose a substantial learning curve. Instead, they'll deliver easy-to-use GUI tools and clear documentation to facilitate rapid successful use of the paradigm without requiring an in-depth knowledge of arcane

marshalling details and new APIs.

Current SOAP Implementations

Most serious SOAP implementations on the market today are Java-based and have leveraged all relevant parts of the Java platform. They are J2EE or, more specifically, servlet applications. A single servlet or JSP, sometimes called a *SOAP listener*, receives incoming SOAP requests. Requests are parsed and dispatched to application logic implemented in local Java classes, remote EJBs, or on other platforms. Some variations generate a unique servlet for each Web service. This servlet and an associated runtime library must then be deployed into the host environment.

The benefits of these approaches include ease of implementation, rapid time-to-market, and a time-tested platform. This architecture, however, suffers from a serious flaw: these SOAP implementations are presented to users as J2EE applications rather than as enterprise platforms for distributed computing. In this context, SOAP message handling is a second-class addition to a platform and not a platform itself. An HTTP servlet-based implementation is subject to the following additional flaws:

- The SOAP implementation architecture is specific to the HTTP protocol. As new protocols are added, users may not be presented with a consistent view of their Web services. It may, for example, require several different steps to activate a single Web service on different protocols. In addition, the interfaces exposed to the Web service implementation are likely to be transport-specific.
- Class version conflicts between individual Web services, the SOAP servlet, dispatchers, and the application server will be difficult and awkward to resolve.
- The deployment model requires that the application logic behind a Web service be visible to the host application server. This often requires that classes and libraries are included in the classpath, which means that the application server must be restarted to deploy a new Web service.
- Web services consist of a collection of configuration data, WSDL, generated code, references to application implementation, and possibly SOAP dispatch code. The developer must correlate these different elements and manually deploy them in a configuration acceptable to the SOAP servlet. This architecture does not provide for bundling these resources to allow reliable deployment of a tested Web service to a production environment.
- Web services require unique management interfaces. Listed alongside 800 deployed servlets, a Web service servlet will not be clearly identified as a unique and powerful resource to be carefully administered.

To overcome these limitations, and to provide a platform for enterprise-level Web services, a new approach is required. A Web service container encapsulates features that help to differentiate a proof-of-concept technology from a platform product.

A Web Service Container

A product capable of supporting projects with the characteristics listed above will provide a first-class environment for deploying and running Web services. A "Web service container" is under development and has three primary characteristics:

- ***Ubiquity:*** Containers will be imbedded within a wide variety of platforms so that a vendor's Web services solution can be made available to virtually any project. Containers will be constructed to reside within an enterprise's existing applications. An analogy might be helpful: the Java virtual machine can be described as a hardware platform. Application servers are operating systems running on this platform. The Web service container is analogous to a network layer that facilitates communication between disparate systems. A critical feature of viable Web service containers is support for many host environments. The Web service container is not simply an extension to J2EE. Hosts will, of course, include major J2EE implementations but containers must also run in enterprise integration servers, message brokers, CORBA servers, and even proprietary environments developed by IT organizations. This ubiquity is necessary if functionality hosted on existing systems is to be exposed as Web services.

  To facilitate rapid evaluation and adoption by potential customers, containers will also be capable of running standalone using lightweight or reference implementation host environments. These standalone versions will likely evolve into full-fledged SOAP Servers. In addition to multiple host environments, the container will support multiple protocols. Most Web service implementations currently support HTTP. Look to see the industry leaders extend implementations to support JMS, SMTP, TCP, and other common delivery vehicles.
- ***Usability:*** Management interfaces and JMX instrumentation will facilitate deployment and administration. Graphical Web service builder tools will facilitate the assembly of Web services from existing functionality. The builder environment will be integrated with the container to facilitate rapid round-trip build, deploy, and test cycles. Savvy vendors will produce tools that are approachable by administrators and analysts who don't have deep systems engineering backgrounds.
- ***Consistency:*** Java-based HTTP SOAP dispatchers are frequently implemented as servlets. They utilize third-party class libraries to manipulate XML, introduce proprietary code, and maintain state within the servlet environment. Functionality to be exposed as a Web service is typically contaminated by these components. Web service containers will isolate application functionality from the container's implementation and host peculiarities. Further more, containers will provide interfaces that allow services to participate in the service life cycle and access the runtime environment. As the container is extended to support additional hosts and transport mechanisms, this environment must be implemented consistently.

Feature Evaluation

As products mature, they'll progress beyond simple servlet implementations toward a container platform. Describing the several possible architectures that can support the platform is beyond the scope of this article. However, specific features will characterize products that are based on sound infrastructure, and teams evaluating Web service products can test for these features. Feature evaluation is a tried-and-true approach to measuring the maturity of an underlying infrastructure. The feature summary below is organized into the familiar taxonomy of install, build, deploy, run, and manage.

Installation

A clean installation process is a hallmark differentiator of products from frameworks and technology previews. Web service containers will be designed to install directly

into running host application servers without changing the host environment. Containers that force XML parser upgrades, mandate specific versions of open source code, or require any changes in the classpath or executable path are unacceptable. IT environments are complicated already and containers can't introduce the possibility of conflicts.

Once installed, the container's functionality will be tightly integrated and appear as a new feature of the host environment. When the host is running, the container is running.

Sophisticated containers will also provide for in-place upgrades to accommodate the rapid evolution of the platform. Users will be able to upgrade the Web service container while it's up, running, and servicing requests. The running services will immediately benefit from a new version's improved performance, scalability, or interoperability. New services can be developed that leverage the new version's additional features.

Build

Web service builders that allow systems analysts to assemble new services will complement containers. Business integration vendors will push toward the ideal no-programming model by delivering builders that support increasingly sophisticated data flow and type mapping specification tools. The builder and container will be tightly integrated to allow rapid round-trip build, deploy, and test cycles. A good test of this integration is the degree to which the builder supports the expression of different mapping choices. Robust containers will allow, for example, a Java Date to be mapped to an xsd:dateTime or let a user extract the year and map it to na xsd:short. An integrated builder will support these container options.

Containers will complement the builder's GUI-based Web service assembly paradigm with programming APIs. Vendors are working to standardize the critical APIs through the JCP process, but don't expect Web service portability any time soon.

When evaluating containers, be wary of systems that offer only part of this picture. Products that attempt a complete no-programming approach and eschew all APIs have no safety net for the inevitable GUI shortcomings. Conversely, products that can't automate the construction of even moderately sophisticated services offer little in developer productivity enhancements and will require extensive Web services expertise.

Deploy

Containers will introduce some form of bundling that couples all materials required for a service. This bundling will likely be accomplished through a zip archive similar to J2EE WARs and EARs. As with the J2EE environment, the purpose of this bundling is to allow for atomic deployment of individual service implementations and to isolate the impact of deploying a new or modified Web service.

Deployment must have no effect on other Web services and other applications running in the host environment. It must never be the case that a newly deployed Web service disrupts operation of the underlying IT application. Under no circumstances should deployment require a restart of the Web services container or the host application server environment. It must be possible to deploy and run a service in a variety of settings without re-bundling. Don't expect Web service archives to be compatible between vendors, but do require that a vendor support the same bundling on all platforms and across all protocols.

Container vendors will provide tools that facilitate assembly and deployment of Web services specified during the build process. Sophisticated implementations will let users simultaneously deploy bundled Web services to containers running in multiple host environments and supporting different transport protocols. An acceptable platform must support live update of running services.

Run

Products that are based on a container infrastructure will support bindings to multiple existing transports. This is a key differentiator. Current HTTP servlet–based technology previews and frameworks aren't designed to support JMS or TCP. Even if your first Web services product doesn't require a fault-tolerant transport, don't make the mistake of choosing a product that won't support these needs in the future. As with any new paradigm, look to companies that are investing in a long-term infrastructure and not flashy demos.

Manage

Deployed Web services will be a critical element of IT operations. They must be managed as first-class elements, not as special purpose servlets. Container-based implementations will support complete management solutions probably based on JMX. Management capabilities will include the ability to activate and inactivate services as well as modify tracing levels and view service statistics. Management will likely be integrated with deployment and enterprise host features like clustering, such that through an administration console, running containers and their associated services can be replicated for scalability and fault tolerance.

Conclusion

No vendor will deliver all of these features in the first releases. But when evaluating platforms, ask questions about the infrastructure. Ensure that your pilot projects succeed not just in constructing a proof of concept, but in evaluating the platform's suitability for the next quarter's deployed system and next year's mission-critical initiative. The Web service container features described above will be difficult to achieve without a suitable architecture. Use them as yardsticks to measure the progress your vendor is making toward a reliable platform.

Inktomi

www.

web services EDGE conference&expo™

The Show Goes On

written by Sean Rhody

THE SOFTWARE INDUSTRY RETURNED TO NEW YORK CITY WITH THE INTERNATIONAL JAVA & WEB SERVICES CONFERENCE AND EXPO AT THE HILTON NEW YORK.

Web Services Edge 2001 East International Web Services Conference & Expo was colocated with the JDJEdge 2001 International Java Developer Conference & Expo

New York, NY, September 23, 2001 - In spite of what many thought might prove insurmountable obstacles, the international software industry has provided New York City today with a resounding indication that heavy hearts and thoughts are not to be permitted to become a barrier to returning to the business of business, including the Internet technology business.

Delegates from various parts of the country and from around the world began gathering at the Hilton New York to attend the leading Java and Web services technology events on the East Coast this year, JDJEdge 2001 International Java™ Developer Conference & Expo and Web Services Edge 2001 International Web Services Conference & Expo East, both produced by SYS-CON Events, Inc. www.sys-con.com.

Coming so soon after the devastating World Trade Center carnage, this is a strong sign that America's software developers and vendors alike are determined to go forward – coming together – to learn, to network, and to do business with each other. The first conference sessions were well attended, in one case so well that there was standing room only.

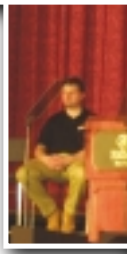
While we were all certainly justified in fearing a light turnout for Web Services Edge in light of the events of September 11, when I walked out to give the opening remarks to a packed room I was proud and grateful that this in fact was not the case. Our first Web Services conference, in conjunction with JDJEdge, was very well attended.

Conference exhibit floor attendees visited exhibitors for the latest in Java and Web Services products and solutions.

The first keynote presentation featured James Gosling, Sun Fellow and Father of Java. Gosling's talk covered a number of topics, including the introduction of Web services. His individual remarks were later amplified by a keynote panel discussion on Web services. The panel consisted of James Gosling; David Litwack, CEO of SilverStream; Dave Chappell, chief evangelist of Sonic Software; Richard Soley, CEO of OMG; Rick Ross, founder of the Java Lobby; Tyler Jewell, BEA evangelist; Don Leclair, VP of Computer Associates; and me.

The Java Architecture Fast Track was one of the sold-out sessions at JDJEdge.

This vocal and opinionated panel covered a variety of topics, including security, discovery, and the very nature of Web services itself. Unfortunately, we didn't get a definition of Web services (or rather, we got eight of them), but we did get some very enlightened insight into what the value proposition of Web services is. It included the ability to finally make J2EE and Microsoft work together, as well as the ability for people to simply, easily do "really cool stuff" with their computers. One of the points of almost total agreement was that Web services is as much a social paradigm shift as a technical one.



The nature of Web services is that people now want to connect their applications, and have agreed on a common approach to doing so. Gosling made the point that a system that inherently relies upon a network of other systems is useless until there is a widespread adoption of such a system. He used the fax machine as an example – what good is one fax machine – and he likened it to the idea of Web services. Not all of the technology for Web services is new, and certainly the idea of connecting applications together has been present for decades. What is compelling is that we have a core set of technologies to do so, we have applications that have been designed in a way that enables swift transition and minimal intrusion, and we have a widespread desire to make our applications communicate.

I believe a key quote from that keynote session was, "Even if we never connect two enterprises together with Web services, it will be a success because we can finally connect to ourselves." Finally, Web services

ESEDGE 2001

Greg Kiessling, CEO of Sitraka Software, was interviewed on SYS-CON Radio by Sean Rhody, editor-in-chief of *Web Services Journal*.

Scott Dietzen, CTO of BEA Systems, built and demonstrated a Web service during his keynote.

will provide interoperability without the vast array of adapters. Joe Menard, CEO of WebGain, referred to this in his session as the M x N problem – for every EAI system (M) and every legacy application (N), there's an M x N order of magnitude of work to be done. Web services removes that by leveling the playing field with dynamic discovery and simple binding.

Discovery and UDDI was a topic on everyone's minds. The panel discussed the ability to create a global DNS as well as the real value in having one particular global system. The prevailing opinion seemed to be that UDDI was destined for smaller, private usage (i.e., a domain per industry) with only limited deployment of services to the public. Part of this was due again to the social nature of the way people do business – they like to know their trading partners. "The idea that Joe's tire shop will provide the next million tires to Ford just because they have a Web service for tires is ridiculous," was one of the more memorable quotes from the panel. People share with other people based on relationships, and any technology that ignores this is doomed.

Later in the week we saw how simple it is to put together a Web service on top of a J2EE server. Scott Dietzen, CTO of BEA's eCommerce Division, stopped for five minutes during his presentation to actually build and demonstrate a Web service. While trivial, it did show the power of the J2EE tools available already, and how they can be exposed as Web services with very little effort. Dietzen actually demonstrated service chaining, combining two Web services to obtain traffic information in French – one to obtain the information, another to translate it from English (à la Babelfish-esque technology). He pointed out how naturally Web services fit on top of the J2EE stack, and how the design principles already employed for J2EE design serve well in the Web services world.

The exposition floor was packed with vendors exhibiting wares aimed at Web services. Cape Clear and Shinka were both present, demonstrating tools aimed at simplification of Web service

Over 1,200 delegates welcomed James Gosling's opening keynote on Monday, September 24.

WEB SERVICES EDGE 2001

web services EDGE conference&expo ™

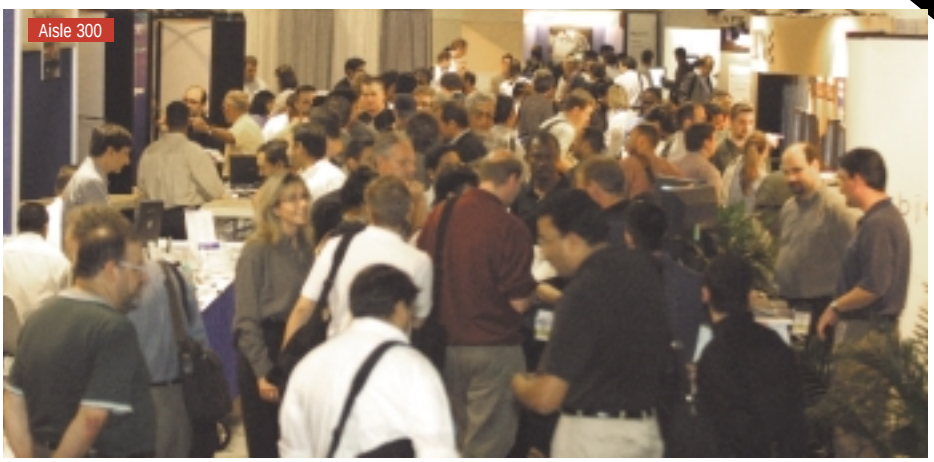
deployment. SilverStream demonstrated their Extend product set, and has made available a Web Services Container free to developers on their Web site. BEA demonstrated the latest version of WebLogic Server, which has been tuned to provide Web service performance from the J2EE stack.

As the show was a mixture of Java and Web services, there were a number of vendors present that were focused primarily on Java, but almost all of the vendors had some take on what a Web service was and how their product line did or would play a part in the world of Web services.

The show closed after three days as a remarkable success. We look forward to seeing you at our next conference. Stay tuned.

Conference registration desk.

Exhibitors made numerous contacts at the Expo.



The largest gathering of Web Services professionals in the industry.

KEYNOTE PANEL: WEB SERVICES PARADIGM

The Web Services Keynote Panel: (front row) Don Leclair, Computer Associates; James Gosling, Sun Microsystems; Richard Soley, OMG; Tyler Jewell, BEA Systems; (back row) Sean Rhody, *Web Services Journal*; Dave Chappell, Sonic Software; David Litwack, SilverStream Software; Rick Ross; Java Lobby.

Watch Web Services Edge 2001 on SYS-CON Television at www.sys-con.com

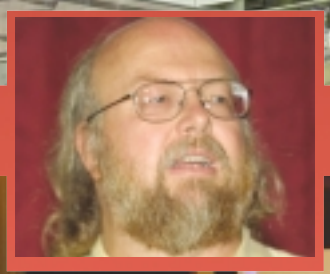
2002

JUNE 24-27

JACOB JAVITS CONVENTION CENTER

NEW YORK, NY

JAVA, XML AND .NET TECHNOLOGIES



James Gosling,
The Father of Java

web services conference&expo EDGE™

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

JDJ conference&expo EDGE™

INTERNATIONAL JAVA DEVELOPER CONFERENCE & EXPO

XML conference&expo EDGE™ 2001

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

WILL OPEN ON NOV. 30, 2001

VISIT WWW.SYS-CON.COM FOR DETAILS

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

Written by Dave Chappell

JAXM: Interoperable SOAP Communications for the Java Platform

Extensibility expedites e-commerce

The Java API for XML Messaging (JAXM) is a new Java application programming interface (API) that provides a standard way for Java applications to send and receive Simple Object Access Protocol (SOAP) messages. The basic idea is to allow developers to spend more time building, sending, receiving, and deconstructing messages for their applications and less time programming low-level XML communications routines. Developed through the Java Community Process, JAXM provides a simple yet flexible standard API for developing and deploying SOAP-based applications that can be truly interoperable with applications developed on other platforms. JAXM will be available as an optional package for the Java 2 platform, Standard Edition and may also be included in future releases of J2SE and Java 2, Enterprise Edition (J2EE).

JAXM is intended to be a messaging API for the development of XML-based business applications that occur primarily, although not exclusively, at the edge of organizations. The need for generating such messages is growing rapidly as corporations begin to do more business over the Internet. What's needed is the ability to communicate not only with a single hub or exchange that is driving the company's move towards e-commerce but with a wide range of business partners. While some hubs such as Wal-Mart and Home Depot have created their own e-commerce standards, the industry as a whole seems to be moving toward the development of industry-wide standards that will expedite e-business interoperability.

SOAP, one of the most important of these new standards, addresses critical transport and packaging issues. SOAP can be thought of as a successor to the XML-RPC standard in that it provides an RPC transport mechanism for XML over HTTP. As stated in the W3C SOAP 1.1 draft: "SOAP is a lightweight protocol, for exchange of information in a decentralized, distributed environment." The SOAP protocol has three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP has received considerable acceptance in the industry as the de facto way of exchanging data between applications in an interoperable fashion.

Lightweight but Adaptable to More Complex Protocols

The minimum requirement to support JAXM is to be able to use a simple set of APIs for sending, receiving, and parsing of SOAP messages. The on-the-wire protocol of a JAXM message is vanilla SOAP over HTTP. Hence a JAXM-enabled application may communicate with any other application that knows how to speak SOAP – including those built on platforms that have nothing to do with Java, such as Microsoft's BizTalk server.

In an all-Java environment the simplest deployment configuration would be to use the Sun reference implementation (RI) as the JAXM "provider." The sending client uses the JAXM APIs to send a SOAP message over HTTP. The receiving client is registered as a servlet, as illustrated in Figure 1.

While the base architecture of JAXM is very

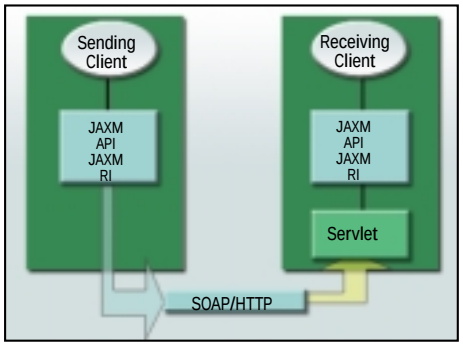


FIGURE 1 **Simple JAXM deployment using HTTP and servlets**

simple, it can be extended to work in a more complex messaging infrastructure such as ebXML through the notion of a base "profile." A JAXM profile is an extension to the JAXM specification that allows a JAXM implementation to take on a certain predefined personality. For example, an ebXML profile would cause a JAXM MessageFactory object to create an ebXML SOAPMessage object instead of a vanilla SOAPMessage object.

The SOAP Packaging

JAXM is based on the SOAP 1.1 and SOAP with Attachments specifications. JAXM offers two different packaging models for SOAP messages, one that includes attachments and one that does not. The SOAPMessage class is the root class for all SOAP messages. If the SOAP message includes AttachmentPart objects, then the message is encoded as a MIME message. A message that contains attachments must have a MIME envelope which contains both the SOAP part

Author Bio

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). Dave has more than 18 years of industry experience in building software tools and infrastructure for application developers, has been published in numerous magazines and spoken at many events.

DCHAPPELL@SONICSOFTWARE.COM

of the message and the MIME attachment. All JAXM clients must be capable of reading SOAP 1.1 messages with attachments.

Creating and Sending Messages

The example below shows how JAXM uses a ProviderConnectionFactory object to create a connection with a messaging provider. The same object can be used later to send the message. The first two lines use the JNDI API to retrieve the appropriate ProviderConnectionFactory object from the naming service where it was registered with MyExchange. This logical name is passed as an argument, then the method lookup returns the ProviderConnectionFactory object that the logical name was bound to. The returned value must be narrowed to a ProviderConnectionFactory object in order to use it to make the connection. A JAXM method is invoked to actually start the connection instance connect1 in the final line of the code.

```
Context ctx = getInitialContext ();
ProviderConnectionFactory cf =
(ProviderConnectionFactory)ctx.lookup("
MyExchange");
ProviderConnection connect1
=cf.createConnection();
```

The MessageFactory class is used to create SOAPMessage objects. A JAXM client may create a new instance of the MessageFactory object for request response messaging using the SOAPConnection newInstance method. On the other hand, when one-way asynchronous messaging is required, the createMessageFactory method of the ProviderConnection object must be used. The following example shows how MessageFactory objects are used to create SOAPMessage objects.

```
MessageFactory MessageFactory1 = con-
nect1.createMessageFactory()
SOAPMessage transact1 =
MessageFactory1.createMessage();
```

JAXM Profiles

It's important to note that SOAP specifies that messages will contain the necessary addressing information but does not define a standard way for this information to be represented within a message. For that reason, JAXM relies on industry-standard usages of SOAP, called *profiles*, for interoperable addressing conventions. For example, the profile may stipulate the specifications of the SOAP header, such as the sender, recipient, message ID and correlation information. The way in which this information is mapped onto a given message is defined by the Profile String. Profiles represent a usage of SOAP by a particular standards group or industry. The getSupportedProfiles()method retrieves a list of the messaging profiles that are supported by the messaging provider to which the ProviderConnection object is connected. For example, ebXML or BizTalk profiles can be built on top of SOAP messaging. ebXML is an emerging standard based on the vision of creating a single global electronic marketplace where companies of any size and geographical location can meet and conduct business through the exchange of XML messages. SOAP is integrated into the ebXML messaging specification in order to provide the underpinnings for its messaging requirements.

> "It's important to note that SOAP specifies that messages will contain the necessary addressing information but does not define a standard way for this information to be represented within a message."

When profiles are being used, an application may not need to specify an address when it sends a message because destination information will be contained in the profile-specific header. If profiles are not used, destinations are specified with an Endpoint object that represents the mapping of a logical name, such as a URI, to a physical location, such as a URL. Typically, an Endpoint object represents a business entity but it may represent a destination of any sort. Conceptually, an Endpoint object is the mapping of a logical name, such as a URI, to a physical location, such as a URL.

In the example above, the method MessageFactory was given no argument so it returned the transact1 object that produces messages using the basic SOAP profile. The code example below shows how to create a MessageFactory object that supports a particular profile. Message objects produced by such a message factory are specific to the named profile. In order to assure interoperability, much scrutiny must be given to the development of profiles.

```
MessageFactory messagefactory2 = con-
nect1.createMessageFactory (<pro-
file1>);
SOAPMessage transact2 = messagefacto-
ry2.createMessage();
```

Multipart Messages

The next step is adding content to the message. SOAP messages are designed to send XML documents in the body of the message. But if the message is to include anything that is not an XML document, then it must have an attachment part. The packaging model for a message without attachments is much simpler, consisting only of a message package with an envelope containing a header and body. A SOAP message with attachments adds the concept of a SOAP part that contains an envelope similar to the one described above. In addition, the message can contain multiple attachment parts that along with the SOAP part are contained in a MIME envelope. There may be any number of attachments and they can contain any type of file.

The following example demonstrates how to build the body of the message. The SOAPMessage contains the SOAPPart, which in turn is used to obtain the SOAPEnvelope. The SOAPEnvelope is then used to obtain the body of the message.

```
SOAPpart spart = transact2.getSOAPPart
();

SOAPEnvelope envelope =
spart.getEnvelope ();
SOAPBody sbody = envelope.getBody();
```

Once the message body is obtained, the SOAPBody.addBodyElement() and SOAPBodyElement.addChildElement() methods are used to create any level of nested elements within the SOAP body, as illustrated in the following code:

```
SOAPBodyElement gltp
= sbody.addBodyElement
(envelope.createName
("GetLastTradePrice",
"ztrade", "http://wombat.ztrade.com"));

gltp.addChildElement
(envelope.createName ("symbol",
 "ztrade",
"http://wombat.ztrade.com")).addTextNode
("PRGS");
```

The following code shows how to create an attachment part. In this example, the content is an image in a GIF file whose url is used to initialize the javax.activation.DataHandler object data handler1. The SOAPMessage object message1 creates the AttachmentPart object attachmentpar1t which is initialized with the data handler containing the url for the image. Finally, attachmentpar1t is added to the message.

> "By providing a standard way to send messages over the Internet from the Java platform, JAXM is almost guaranteed a successful launch."

```
URL url = new URL("http://main/pic-
ture.gif");
DataHandler datahandler1 = new
DataHandler(url);
AttachmentPart attachmentpart1 = mes-
sage1.createAttachmentPart(datahan-
dler1);
message1.addAttachmentPart(attachment-
part1);
```

The last step to be covered here is sending the message using a Provider Connection object which, as described earlier, is a connection to a messaging provider. If asynchronous messaging is being done in the context of a container, such as a servlet or J2EE container, the send method is used. The send method sends the given SOAPMessage object and returns immediately after handing the message over to the messaging provider. No assumptions are made regarding the ultimate success or failure of message delivery at the time this method returns. The ProviderConnection send method requires one parameter, the SOAPMessage object that's being sent.

```
SOAPMessage reply =
connect1.send(transact1);
```

The call method on the SOAPConnection object is used to send a synchronous message. The call method will block until a SOAPMessage object is received. A JAXM application may use the call method to implement the client side of a simple point-to-point synchronous one-way message exchange. Two parameters are required for the call method, the SOAPMessage object that is being sent and the Endpoint object representing the destination. For point-to-point plain SOAP messaging, where no profile is used, an application must supply an Endpoint object to the call method to indicate the intended destination of the message. The default identification for an Endpoint object is a URI. At a minimum, this is what JAXM messaging providers are required to support to identify a destination. Endpoint objects can be created using the constructor, or they can be looked up in a naming scheme. The Endpoint subclass URLEndpoint can be used to send a message directly to a remote party without using a messaging provider.

The following code example creates an Endpoint object called endpoint1 from a URL for the intended recipient and passes it along with the SOAPMessage object message1 to the SOAPConnection call method, creating a synchronous transmission.

```
Endpoint endpoint1 = new Endpoint
("http://myexchange.com/transactions");
con.call(message1, endpoint1);
```

The JAXM 'Provider'

In lieu of actually supplying the more detailed profiles, the base API is very light on "Messaging" and heavy on SOAP content. The majority of the API is built around how to construct and deconstruct SOAP messages. However, there is much allusion to the JAXM "provider" woven throughout the specification.

In distributed application communications among enterprises and across business entities, the use of SOAP over vanilla HTTP just doesn't cut it. Parties are often unreachable, yet reliability of communications is crucial. This gap may be filled in the future by more powerful JAXM messaging providers based on protocols, such as JMS, that provide a simple, robust way of addressing these issues. The JMS specification offers a robust messaging model that supports a variety of asynchronous and synchronous communication mechanisms.

JMS comprises an API and semantics for a middleware messaging system that helps insulate the application from the issues posed by loosely coupled systems by providing services, such as persistence, verification, and transaction support. It has all the semantics of failure scenarios built in via a loosely coupled disconnected operation, persistent messaging, internal acknowledgment of message receipt and rules of engagement regarding transactional recovery and message redelivery. The very minimum required for the two technologies (JAXM and JMS) to work together is the ability for a JMS provider to be able to bridge to SOAP/HTTP.

By providing a standard way to send messages over the Internet from the Java platform, JAXM is almost guaranteed a successful launch. This new standard will provide a major impetus to the creation of many-to-many B2B e-commerce applications using industry standard technology. Perhaps the most important feature of JAXM is its extensibility – its messaging provider and profile concepts will allow it to easily support new interoperability concepts in the future.

Shinka

www.

Written by Kurt Cagle

The Real Niche for Web Services Part 2

Tying up the loose pieces of evolving systems

Last month, in Part 1 of this article, I cautioned about the potential invasiveness of Web services. It's a scary thought that companies could have that much personal information about their customers, but I added then that there are some advantages to Web services, especially in the area of business-to-business. This month I focus on these advantages.

The Last Gold Rush

Business-to-business, or B2B, may have been the last gold rush of the 20th century. And like the gold rushes of yore, usually the guy selling the shovels is the only one making the real profit.

Web services would be perfect in the B2B arena, right? At least, that's what all the e-commerce companies are saying.

The idea for B2B is simple. A significant number of the transactions that occur in business take place between the boundaries of companies. A purchase order is sent in, a shipping transaction is made, and an invoice is sent out.

Each of these transactions, crossing enterprise boundaries, involves the passing of a piece of paper and the time and work to process that paper. By eliminating the paper (and incidentally many of the people originally involved in handling the paper), you significantly reduce both the time and cost of each transaction. Of course, that was obvious back in the 1970s.

After a lengthy and complex process, the UN came up with the Electronic Data Interchange (EDI) standard; a highly efficient system of describing most of the standard documents used in commercial ventures. Of course, the same issue of efficiency versus flexibility reared its ugly head here – and many companies discovered that there were always specific EDI fields that were either missing or not appropriate to their particular circumstances.

Semantic incompatibility rapidly emerged between the standards, and soon a cottage industry of converting from one EDI format to another became the foundation for many of the largest software companies in the world.

These Value Added Networks (VANs) performed a critical service – and charged accordingly. In all fairness, the VANs often did provide ancillary data services as well, but there is no question that the core business of rectifying incompatibilities between different supposedly standard documents remained their primary bread and butter until the late 1990s.

Hello XML

When XML first emerged, it was meant to solve a different problem: providing a framework for describing different forms of documentation. It was not, initially, meant to be either an e-commerce or data solution. But XML's ability to model a wide number of data objects raised the possibility that it could in fact serve as the right wedge for a number of upstarts to get into the same business as the old guard VANs. To a certain extent it was also seen by some leaders in other industries as a way to eliminate the costs of the VANs from their transactions.

I remember watching with some amusement in the late 1990s as these companies raced to either declare standards for their industry as a whole or raced to become the repository of all of the schemas that everyone was declaring. What happened instead was the realization that (even with languages such as XSLT to handle the sometimes thorny problem of translating syntax) most commercial documents are semantically different from one another.

In some arenas, where one or two large companies effectively dominated the landscape, e-commerce systems emerged dedicated to that particular industry vertical. The automobile and aerospace industries made significant gains in setting uniform standards, and in both cases you were looking at one company that dominated enough of the field that they could push standards down the pipe to their suppliers and up the pipe to their distributors. These companies, however, also had fairly robust EDI solutions in place as well, and they had the deep pockets necessary to test a number of different solutions and eventually choose the best ones.

The incarnation of Web services in this arena basically works along the scenario that a company can write an application submitting purchase orders to the company they buy from through the use of a Web service. You could then send in one or more purchase orders to the company in question (or potentially to a company that the other company outsourced to handle its Web services interactions – the new VANs) and get your order fulfilled in the space of seconds, not days.

Of course, if that wasn't enough, the next logical stage was also offered by companies such as Microsoft, IBM, and Ariba – the Universal Discovery, Description, and Integration language (UDDI). This particular XML-based standard proposal works on the premise that there are one or more UDDI databases that provide information about companies, in the order of White Pages, Yellow Pages, and Green Pages.

- The White Pages standard provides critical contact information for people who work at a given company (think LDAP information data here) as well as a directory of public servers.
- The Yellow Pages contain a listing of the company in various taxonomies – what field it's in, what its primary services are.
- The Green Pages provide direct access to the Web services that the company supports.

Author Bio

Kurt Cagle is an author and developer specializing in XML, XSLT, and web technologies issues. He is the president of Cagle Communications, located in Olympia, Washington, and he can be reached either via his Web site at http://www.kurtcagle.net or by KURT@KURTCAGLE.NET

UDDI is still in its infancy, but it's interesting to note that while it

has garnered a great deal of interest, most of that interest comes from companies in the computer field that are interested in building services around such a directory. The response from most other industries has been deafening silence.

A Superfluous Service

UDDI to me represents a significant part of the reason why Web services will not be adopted for several years (and maybe a few decades). It's a solution to the software provider's problem of providing services in this so-called services economy – become a phone book for someone who culls information from that phone book. Of course, if companies already have a perfectly good phone book (and most do), then what's being offered is largely superfluous.

Consider this: a phone book is an advertising vehicle. If you're looking for a certain type of vendor, you open up the yellow pages and see the listings of all of the providers of a given type, listed in alphabetical (or perhaps first-come–first-served) order. You would also see that some ads are bigger than others and some have two- or three- or four-color pictures and large type while others are no more than a name and a number. Being human, you would most likely choose the one with the biggest ad.

UDDI, on the other hand, gives you all of the companies that have signed up in that taxonomy. You might use other UDDI fields to narrow the choice a bit, but that also may exclude companies that provide similar services or products or that didn't end up in the right taxonomy.

But what is perhaps most important to a company is that you don't simply become one of all the companies that provide the service in Seattle, Washington; you become one of all the companies that provide the service in the world. It's a highly efficient solution from a programming standpoint, but from the standpoint of a company's marketing manager, it's absolute suicide – no advertising, no real way of differentiating yourself, no game plan. In fact, chances of being selected by someone wanting your wares is largely dependent upon your place in the hierarchy.

Some retrenchment has occurred in this position. Now UDDI is being touted as a way (assuming you've already selected a company to work with) to know which Web services they have as well as to provide connections to key people in the organization. This last point of course is a bonanza for headhunters, corporate snoops, and anyone who may be disaffected with the company – not to mention breaching the social firewall that a well-trained receptionist usually provides for a company.

This gets back to the Green Pages, which in many respects are the only pieces of UDDI worth considering, from a corporate standpoint, though again not necessarily for the reasons that are promoted.

One useful way of thinking about a Web service is that it's an API function for a specific server. The aggregate of all the Web services on all of the host servers in a company essentially makes the company into a document object model. Thus, you could effectively createmy Company.financial. purchaseOrder with the specific service myCompany.financ ial. purchaseOrder. send( ). The Green Pages provide descriptions to the companies' Web services, including the expected parameters and result sets.

> "Web services represent… a profound shift because a Web services system can change incrementally."

This information can be quite useful, but the creation of a UDDI infrastructure that performs this actually adds an additional layer that could be solved just as easily by an e-mail or phone call (Hey Joe, send me the link to your WSDL – Fred). In many respects it's far more secure than posting this information in open repositories, and makes it easier to keep such a system up-to-date.

The Intrabusiness Solution

I think that business-to-business Web services solutions will eventually come but not for a number of years. The principal reason for this is that currently the B2B approach to Web services seems arbitrarily tacked onto the existing business infrastructure. It requires a great deal of cooperation between people who ordinarily don't even talk to one another, pre-empts many major roles in a company (marketing, sales, even management solutions), and exposes companies to the dual possibility of buggy software and deliberate hacking. Finally, in this day and age of layoffs and companies paring to the bone, the adoption, integration, and management of such external points of contact are hard to justify when existing systems work adequately and the cost of the new systems (in terms of developer time and potential disruption) is simply too great.

Yet I would argue that there are three areas where Web services make perfect sense: intrabusiness APIs, communication systems, and device-to-device telematics. None of these are being pushed as sexy solutions, in great part because the boundaries they cross don't offer the potential to make a profit – though they can significantly reduce costs.

Intrabusiness APIs First

How many computer applications does a typical company have? Even a small company probably has dozens – between commercial applications like word processing programs and internal applications developed to fulfill specific needs within the company. Some of these applications are finding their way to portals, single points of contact within a company, but in many cases people need access to the data directly rather than to someone else's view of that data.

Suppose for a moment that you have several databases with different types of data that can be extracted for use. A typical developer needs to determine where that data is, move it into a convenient form for processing, then write the requisite code for manipulating that data.

In a small shop, it's likely that one person has written most of the applications and can generally tell you when duplications occur or where some piece of software mirrors something that already exists. On the other hand, if you have a company with distinct facilities and distributed IT departments, then in all likelihood you'll run into situations where the same or similar tools are developed (sometimes repeatedly); where

multiple versions of tools are in circulation at any given time; and where incompatibilities and gridlock soon develop. This is especially true in places that develop class libraries. Such incompatibilities can be both costly and frustrating to resolve.

Web services actually work ideally here. Consider the case of Acme Widgets. They have run into the problems described earlier, and in the wake of layoffs are now struggling with too many applications that are either unsupported or suffer from incompatibilities between versions. The IT manager, Sheila Jenkins, looks at the company's needs and sets up a set of Web services APIs that developers can start referencing in their own work. The services include version information and multiple versions of each API are maintained as they are developed. This means that a programmer writing against the API needs only to specify the version in a Web services call to ensure that the code continues working even as the system evolves.

In addition, applications written against these services are themselves designed as Web services that run on local servers. On any given day, Jennie Martin, a programmer in R&D, writes an application for tracking progress of research efforts and makes information available in a wide variety of forms. The program works well and becomes popular. In a review of local services, Sheila decides to promote Jennie's application to the level of a company-wide API. A new version of the program is set up, and older versions that ping against the local servers basically do nothing but link to the more recent versions of the API on the company servers.

An Organic API

Over time, then, an organic API develops for the company. The limitations of versioning actually become an advantage, and increasingly the code in a company becomes its own; assets that in turn could be sold to other companies as a product. Documentation becomes simpler as well, as any API version would be designed to include its own documentation.

Finally, because Web services are effectively agnostic, Web services could be developed in any language – C++, VB, Java, JavaScript, XSLT, Perl. It doesn't really matter. This is not due to any deep IDL or other "magic" technology. Rather, a Web service is simply an abstraction, an interface, into the actual API set.

The fact that such Web-service implementations are now appearing for most languages (including several that are nonproprietary or Open Source) effectively makes it possible to achieve that holy grail of programming: language-independent code. It does this pretty much by default; not by enforcing a single code base as Java does but by rendering the interfaces in XML and the implementation in the language of choice. This isn't all that different from the way that .NET operates, save that Microsoft approaches the problem by creating an intermediate IDL from each language, then compiles that.

> "…there are three areas where Web services make perfect sense: intrabusiness APIs, communication systems, and device-to-device telematics."

The idea of Web services as the basis of organic, internal APIs fits well into the XML paradigm as well. XML has this curious characteristic anyway: once it finds its way into a system, it tends over time to become a pervasive part of that system. I think that's because XML is an abstraction layer that makes the interconnections between data, code, and documentation become far more obvious than they tend to be in a procedural, object-oriented world. Web services similarly abstracts not just the implementation but also the language and connection points of the interface, turning API calls into Web addresses.

The Irony of Web Services

The irony of an organic intracompany Web services package is that it will, in the end, provide the basis for a true business-to-business Web services environment.

One of the principal problems with Web services is security, which is generally less of an issue on an internal network. Once the intracompany Web services network is created, there's no reason that a separate set of APIs couldn't be written that simply wrap the existing APIs in a stronger security layer. Thus, rather than spending huge amounts of money and resources now trying to tack on yet another set of commercial interfaces into an already stressed system, organic intranets let outside users leverage the unique expertise of the company's tools and resources at a much lower cost in an incremental fashion.

This last point can't be stressed enough. There are some deep systemic problems with the way software is created now because it is still seen largely as a product that must be presented all at once. In my experience, however, software grows within a company: new programs are added to solve problems, older programs get archived to handle legacy systems, periodically a program is killed, and a system is upgraded when the cost of maintenance becomes too high to justify continued use of the project. Occasionally you get systems management consultants who come in and attempt to impose top-down re-architecture, but not surprisingly most of these initiatives end up failing because this approach neglects the fact that software evolves in response to need.

Web services in that regard represent a profound shift because a Web services system can change incrementally. For example, consider a document-management system based on Web services. The system is Web-based, with a browser for a front end that configures its menus and other options based upon a call to a Web service. One night, a Web services administrator adds support for a new document type to the system, creates an incremental version change to the Web services that support the file chooser, and notifies the master versioning system that a new version has been added. When an editor opens up the application in her browser the next morning, a listing of all changes that took place throughout the night appear in the initial splash screen – and when she goes to open files she discovers that she can now read, edit, and write the new document format.

This transparency means that a basic system can be created and then, as people respond to the application, it can be shaped to more accurately represent the requirements for the app. Is a particular feature especially disliked?

The feature can be made to go away, but, because of versioning, people who found they preferred the older system can choose to work on earlier versions. Note that this is obviously not perfect – you will undoubtedly run into situations where a user has to choose between a feature that they liked and working with functionality that a newer version added. But compared to the current deployment hell that most systems administrators run into every day, a problem like this would be fairly minor.

The one area this approach stresses is user education: new features take time to learn. If, however, one of the requirements for promoting a feature to a production-level server is the creation of adequate documentation for that feature, then you can simultaneously solve the problem of documenting the application as a whole (by distributing the writing of such documenting) and of ensuring that features are sufficiently constructed so that documentation can be written for them.

Moreover, currently upgrading to new software packages often involves learning significantly different ways of doing things, not to mention all of the bells and whistles that were added for marketing purposes but that users need to evaluate for their own needs. Incremental development and deployment minimize this.

I think this particular use of Web services will ultimately become one of two dominant ones. It's instructive to note that for all the hype that many of the larger software vendors make about Web services being destined for B2B systems and interchanges, developers who are creating pilot and test projects using Web services are finding they are increasingly their own best customers of these services. Organic development is already under way in a lot of companies without admitting that this is precisely what's happening. Time will tell.

Peering into Peer-To-Peer

There is another aspect to Web services that will likely emerge in the same organic, subtle way, though it will take longer for it to happen.

The Internet has two operating models at the moment. The first was a major reason the Web exploded in the first place: when Tim Berners-Lee wrote the first Web browser, he also, simultaneously, wrote the first Web server. At the time, these two pieces of software were seen as analogous to a short-wave radio – the server sent messages out while the browser received them. So long as they were in the same box, it was easy to see the nascent Web as a collection of peers.

In the early 1990s, however, as the Web exploded, the demand to access the Web exceeded the computational and network capabilities that existed. More and more people accessed the Web through dial-up accounts because the cost of owning a T-1 (or even a fractional T-1) was so high.

This discrepancy turned the Web into a hub-and-spoke system where large ISPs invested heavily in infrastructure to support the biggest bandwidth connections. They then leased these to intermediate providers who made the services available to people with slow dial-up connections. The Web became strongly hierarchical, which meant that someone wanting to set up a Web site on their own machine was serving it at a thousandth of the speed that the most powerful servers were sending information.

The Second Phenomenon

A second, related phenomenon has been the effort on the part of service providers to restrict upload access to the Web. In some places this involved a cap on uplink speeds, in some places it meant the creation of DHCP that would efficiently allocate increasingly rare DSN numbers. Ipv6 is intended to solve this latter problem, but for at least some server vendors, a move to Ipv6 will likely take some time.

In other words, an increasing asymmetry between those who provide content and those who consume it has been occurring. The current vision of Web services is conceived to strengthen this, as it places a significant premium on

the server to provide the applications involved. Microsoft's Hailstorm is in fact a strong example of this mindset in that, while touted as being a distributed service, it will place a significant amount of the utilization of personal Web services in one of the largest client/server systems the world has ever seen.

Yet, much like the organic development of Web services as intrabusiness applications, there are some intriguing hints that Web services may ultimately lead to the systematic decentralization of the Web, though it will be a process that may take years to play out. To understand why, consider again what a Web services system is. Ultimately, it's a set of APIs that abstract the server as a semantic provider of information. Thus, you can talk about a "server" labeled as www.mycompany.com/finance that provides a set of finance-oriented methods. This is the big-box view of servers.

But more and more you're seeing cell phones, PDAs, and personal laptops with wireless connections become the dominant way of connecting to the Web, and these devices pose two problems to those who prefer to see centralization of services. The first is that they're increasingly connected to the Web via always-on connections, which in turn means that dynamic IP allocation can't be used – you need absolute address.

The second is that a Web services model actually works better with these devices than the traditional high bandwidth HTML that has been the staple of Web pages. Most handhelds (as well as toys, a subject I'll bring up again) are from a programming standpoint simple and well-defined APIs; far simpler in fact than the rich API set of even the most basic general server.

Helping Bedeviled Programmers

Encoding a Web services stack into these devices could bypass many of the incompatibility and versioning issues that currently bedevil programmers working within that sphere. Such servers would be simple and relatively static, but communicating with any mobile device often requires only a few methods anyway. Moreover, it may be possible to put such services interfaces into software that could be updated through a Web service itself, so that such devices are always using the latest API. Thus, each of these devices becomes www.myPDA.phone/personal or something similar. (Actually, most such devices would just be referred to by their Ipv6 address, but there's no reason why a named address couldn't work here.)

This in turn works its way up the chain. I have a laptop with a wireless Ethernet connection that turns it into a roving server. I have a second dedicated desktop server with a services connection set up that periodically queries the laptop – if it's live (and I give my permission), then any future queries get forwarded to my laptop until I disconnect.

> "…there are some intriguing hints that Web services may ultimately lead to the systematic decentralization of the Web, though it will be a process that may take years to play out."

"Client-side" Web services are much more likely to be interface-oriented, by the way, while "server-side" Web services are more typically going to be data-centric, but beyond these distinctions the real boundaries between client and server pretty much disappear.

This distinction is also disappearing in dedicated chat systems. If your system hosts a set of Web service APIs that are dedicated to a chat application, then subscribers to the same set of protocols who had (or could discover) your IP address could communicate directly with you, without connecting into a central server.

Underground Systems

These systems would start out as largely "underground" because for those who currently are trying to profit from the Web, the only real way to do it is to become the intermediary who brokers the interactions between people (by charging a fee for either access to the network or for each transaction). Yet, in time, perhaps three to five years, this form of specialized chat (or file sharing, which SOAP would be ideal for) will probably render most instant message and similar services in existence today obsolete.

I think in the long term these Web services will also render our current browsers to a quaint, curiosity status. A browser is in effect an early Web service client: it makes a request to a server for a set of data that happens to be a document of some sort (or a media object), and then has the intelligence to convert the response into a workable interface called a Web page.

As XML and XSLT become more prevalent, the role of the browser becomes subsumed by the operating system; if you can define an interface via XML (it doesn't have to be XHTML, by the way – just the instructions that tell where specific widgets are placed, what hooks they have within them to data, and how these widgets fit with other services), then the whole notion of interface programming over time becomes a function of XML-based services. Already, you see this with XUL, the language that Mozilla uses to describe its own interfaces.

In many ways, this is what I see as the true personal Web services – the abstraction of interfaces, the decentralization of applications, transient networks built upon protocols that themselves can shift and change in response to requirements, the movement away from large, concentrated systems and toward systems that encourage what the Internet was originally designed for: communication between people.

Missed an issue?

We've got 'em all for you on CD!

JAVA DEVELOPER'S JOURNAL

The most complete library of exclusive *JDJ* articles on one CD!

Check out over 500 articles covering topics such as...

Java Fundamentals, Advanced Java, Object Orientation, Java Applets, AWT, Swing, Threads, JavaBeans, Java & Databases, Security, Client/Server, Java Servlets, Server Side, Enterprise Java, Java Native Interface, CORBA, Libraries, Embedded Java, XML, Wireless, IDEs, and much more!

JDJ The Complete Works
~~Reg $119.99~~
Buy Online
Only $71.99

XML JOURNAL

The most complete library of exclusive *XML-J* articles on one CD!

Check out over 150 articles covering topics such as...

XML in Transit, XML B2B, Java & XML, The XML Files, XML & WML, Voice XML, SYS-CON Radio, XML & XSLT, XML & XSL, XML & XHTML, 2B or Not 2B, XML Industry Insider, XML Script, <e-BizML>, XML & Business, XML Demystified, XML & E-Commerce, XML Middleware, and much more!

XML-J The Complete Works
~~Reg $59.99~~
Buy Online
Only $53.99

COLDFUSION Developer's Journal

The most complete library of exclusive *CFDJ* articles!

Check out over 250 articles covering topics such as...

Custom Tags, ColdFusion and Java, Finding a Web Host, Conference Reports, Server Stability, Site Performance, SYS-CON Radio, ColdFusion Tips and Techniques, Using XML and XSLT with ColdFusion, Fusebox, Building E-Business Apps, Applicaton Frameworks, Error Handling, and more!

CFDJ The Complete Works
~~Reg $79.99~~
Buy Online
Only $71.99

CF Advisor

The most complete library of exclusive *CFA* articles!

Check out over 200 articles covering topics such as...

E-Commerce, Interviews, Custom Tags, Fusebox, Editorials, Databases, News, CF & Java, CFBasics, Reviews, Scalability, Enterprise CF, CF Applications, CF Tips & Techniques, Programming Techniques, Forms, Object-Oriented CF, WDDX, Upgrading CF, Programming Tips, Wireless, Verity, Source Code, and more!

CFA The Complete Works
~~Reg $79.99~~
Buy Online
Only $71.99

SPECIAL OFFER:
Buy CFDJ & CFA The Complete Works
For Only $129.99

JDJStore.com

Order Online and Save 10% or More!

www.JDJSTORE.com

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

COLLECT ALL 4 FOR ONLY $229.99
JDJ, XML-J CFDJ & CFA

Atoms vs Bits and the Digital Middle Mile

Agreement on standards expedites sharing among partners

Written by Noel W. Clarke

I believe that Web services may make the long-standing battle of open source software (OSS) versus closed source software (CSS) almost irrelevant. What!, you cry in incredulity. He can't mean that! How can he discount the efforts and thinking of some of the world's most brilliant programmers? However, before you blast out a scathing e-mail flaming me to a crisp, listen to my reasons, then decide if you don't agree with me.

The Middle-Mile Solution

Recently I've come to think of Web services as a "Middle-Mile" solution. To understand why, it's helpful to look at a simple example – the steaming cup of tea on my desk. The essential players in getting my tea in the morning are the grower/manufacturer, the distributor, the local store, and, of course, me – the end user. The transportation of the tea leaves or "raw goods" from the manufacturer to the distributor is the figurative "First Mile." The distance from the distributor to the local store is the "Middle Mile," and my walk to the store to get my tea leaves is the "Last Mile." The trains and trucks that form this middle mile by transporting tea leaves to my local store rely on customers like me to consume the tea leaves when we come into the store to buy the tea. Likewise, I rely on them to transport tea leaves to my local store so they're ready and waiting when I want tea.

Web services provide the solution equivalent to the trucks and trains in my tea example – they move the goods from the manufacturer/distributor to the local store/end user. However, rather than a physical "hard good" such as tea leaves, Web services instead transport an electronic "soft good." In this way, Web services are the middle-mile solution for soft goods – dealing with bits, not atoms.

These electronic soft goods are usually small pieces of a business's core logic, exposed so they can be called by a business partner, customer, employee, or another application. Almost every business has discrete shareable business functions, ones that are often core to a particular business interaction with its partners. In addition, almost every business wants to use shareable business functions from their partners, to exchange information surrounding commerce. What's needed is agreement on the actual mechanism and standards – enter Web services.

Standards for the Infrastructure

Web services define standards for the infrastructure to remotely call discrete shareable business functions. This isn't new. We've had the ability to invoke remote objects since our applications got bigger and we started to spread them around – moving to three-tier and then to *n*-tier architectures. However, in the case of Web services, it's done with protocols and data representations that are ubiquitous and Internet-friendly. The wire protocol is usually HTTP over TCP/IP but others are allowed (such as SMTP); the network data representation is XML; the interface definition language is WSDL (an application of XML); and the registry for Web services is UDDI. All of these standards are blessed and espoused by a collection of vendors the likes of which we've never seen before, including IBM, Microsoft, SAP, Compaq, Ariba, and many more.

HTTP, WSDL xSchema, and UDDI are the current offerings but other protocols, data encodings, interface definitions, and registries could become popular and they should be easily pluggable. The developer requirements to call a Web service are made trivial by the use of a SOAP ORB. This is analogous in function to a CORBA ORB in that it allows the developer to call methods on remote objects as if they were local objects without regard for the underlying mechanism. This includes support for strongly typed objects into and out of the method calls, which implies support for standard and customizable data marshallers/de-marshallers.

Implementations of almost all of the basic technological requirements are already available as OSS projects, Apache SOAP for example. Additional portions of the "Web Services Stack" are currently under development in ongoing OSS efforts – Apache AXIS (a better SOAP ORB), WSDL, UDDI. These efforts allow value to be added from the tools front. These tools should strive to make it easy for developers to wire together services into business process flows, and then to call these flows – otherwise known as applications.

Web services enable these next generation applications, which are really composed of software accessible across the Internet by common protocols, and open standards. These applications travel the "middle mile" between the trading partners and allow meaningful e-business conversations, with or without human involvement. Like open source software, Web services rely on the "network effect" Eric S. Raymond (author of the seminal works on open source software development including *The Cathedral & the Bazaar*, "Homesteading the Noosphere," and "The Magic Cauldron") describes – however, in a slightly different way. While Web services can be used between applications inside the same corporate network, or across a LAN between divisions, their value is fully realized when connecting trading partners.

Author Bio

Noel W. Clarke is a senior e-business strategist for SilverStream Software Inc., where he's also held the positions of international technical account manager and product marketing manager for the B2B Division. Born in Durban, South Africa, he holds a bachelor of science degree in molecular biology from the University of Calgary.
NOEL@SILVERSTREAM.COM

The developer metaphor for invoking a Web service is very simple in theory and practice. First there's the matter of gen-

erating a stub from the WSDL file that describes the service: where it is, what it requires, and what it returns. Once this stub is created the developer uses it to call the remote code. This stub can also be used in larger constructs to wire together whole modules, sub-systems, or indeed whole processes. All this can be irrespective of what language any of the subtended Web services are written in, what operating system they're running on, or even where they physically execute.

The real value of Web services is their black box functionality. They can be used across devices, cross-platform, and cross-vendor. All any developer has to know is the location to the WSDL file and they can use the logic embedded in the Web service. The developer specifically does *not* own the execution environment.

Open Source vs Closed Source

Now that we know about Web services, let's return to the question of open source software versus closed source software. Most of you are probably familiar with object-oriented programming and have built applications on these principles. Two camps of thinking have emerged: the "Open Source Software" camp and the "Closed Source Software" camp.

Raymond, in his article "Homesteading the Noosphere," outlines some of the breadth and depth of the greatly varying "hacker" opinions. Among the zealous hackers he says that attitudes range from, "Free software is my life! I exist to create useful, beautiful programs and information resources, and then give them away," to, "Yes, open source is OK sometimes. I play with it and respect people who build it." And from the anti-commercial hackers the sentiments range from "Commercial software is theft and hoarding. I write free software to end this evil," to "Commercial software is fine, as long as I get the source or it does what I want it to do."

In his recent work "The Magic Cauldron," Raymond stated that when developing software, the following discriminators push projects towards open source:

- Reliability/stability/scalability are critical.
- Correctness of design and implementation can't readily be verified by means other than independent peer review.
- Software is critical to the user's control of his/her business.
- Software establishes or enables a common computing and communications infrastructure.
- Key methods (or functional equivalents of them) are part of common engineering knowledge.

Let's examine each of these with respect to their application to Web services between business partners. Remember, our business has its core competency, and we rely on business partners for parts of our business beyond our core competence.

Web services allow us to call the most critical business functions to our business; however, these might reside on systems and infrastructure that are owned and operated by our business partners. Our partners own these mission-critical core business functions. So, if these services don't work when they're called, your natural response will be to find another partner, not to try to run the code yourself. After all, you can't do everything, that's why you have partners.

This business function will either be something quite simple and well understood like an order entry system or a product catalog, or it will be insanely complex and very proprietary. In the first case peer review isn't necessary; in the latter your partner isn't going to open the source – this is their core business value-add.

Applications built with various Web services will be critical to your business. However, they're owned by your partner, and so the selection of partner is critical – again open or closed source doesn't matter – if the partner isn't there, the service won't be there.

The entire Web services stack relies on a common communications infrastructure – the Internet. However, the implementation can be called irrespective of the computing platform used. A common communications infrastructure is used between Web services – and much of this is already open source. But the code that's used by a Web service is a different matter.

Even though the process of a given business function is well understood, it's *not* in your business or your sphere of competency, and therefore you may use someone else's service but probably wouldn't be inclined to run the code in-house. Indeed, one of the benefits of sharing Web services is in not owning the implementation, which means you have no knowledge of the source.

Conclusion

So, whether your business processes are coded in open source or closed source software seems almost irrelevant when using Web services. Web services de-couple systems and abstract their interfaces through XML and HTTP. As such, they're capable of bridging the acrimonious chasm between the operating system vendors and may be powerful enough to hurdle the rift between the open source and closed source communities. In the end, you don't have to know the implementation details of a particular business function. All you need to know is that it works. If it doesn't, it's your business partner's responsibility to fix it – or it's your option to find another partner.

Understanding UDDI tModels and Taxonomies

The key integration point for products supporting Web services

Written by Andy Grove

The Universal Description, Discovery, and Integration (UDDI) specifications provide a standardized way for businesses to discover each other over the Internet and find details of services provided. UDDI is a combination of White Pages (contacts, addresses, telephone numbers), Yellow Pages (listing by industry classification), and Green Pages (technical details of services offered). UDDI is indeed universal in scope, in that the services offered can be anything from a telephone number to a SOAP endpoint for a Web service.

Businesses that want to publish their details on the Internet can use either a Web interface provided by one of the public UDDI operators (currently IBM and Microsoft), or a programmatic interface from within their IT systems. Information published to one public UDDI repository becomes replicated to the other public repositories automatically.

UDDI also has great value behind the firewall in intranet environments. There are advantages to using Web services to connect internal systems, rather than implementation-specific protocols, such as RMI, CORBA, or DCOM. Web services are truly independent of vendor, platform, and language, and can minimize the pain of integration issues. An internal, private UDDI repository can then be used as a logically centralized database of these systems. This repository can be replicated across sites within the company using the same techniques as public UDDI repositories.

TABLE 1: tModel meaning by context

Relationship	tModel meaning
businessEntity - category	Classification taxonomy, such as NAICS
businessEntity - identifier	Identification taxonomy, such as D-U-N-S number
businessService - category	Classification taxonomy
bindingTemplate	Technical specification (Service Type) of the service being offered

Web Services Fundamentals

Before we go into the details behind UDDI, it's important to have an understanding of two other standards – SOAP and WSDL.

The Simple Object Access Protocol (SOAP) specification provides an XML-based dialect for invoking Web services across Internet or intranet environments. The specification details a particular XML structure for making a request, including detailed rules on encoding different data types to ensure that SOAP implementations from different vendors are compatible. SOAP defines the structure of the message but, contrary to its name, isn't really a protocol. Instead, SOAP is usable with existing Internet protocols, such as HTTP and SMTP.

The Web Service Description Language (WSDL) is another XML-based specification that provides a format for describing the interface to a Web service. The WSDL file lists the operations, parameters and data types used by that service and is all a developer needs to write a client application that can use that Web service.

Although WSDL documents describe a programmatic interface to a Web service, they're not enough to enable Web services to be published or discovered over the Internet. UDDI is roughly equivalent to a DNS for Web services, allowing companies to change details about where services are hosted. Applications connecting to those services should always locate the service through UDDI first.

UDDI Basics

The UDDI specification provides four core XML data structures to describe a business and the services it offers:

- businessEntity
- businessService
- bindingTemplate
- tModel

The first three data structures are simple to understand. A businessEntity structure provides information about a business, such as the head office address. Each business entity can have multiple businessService structures associated with it – one for each service offered by that business. Each business service can have multiple bindingTemplate entries. The binding template represents the way that the business service is accessed. For instance, a binding template can represent a telephone number, a Web site, or a Web service.

Introducing tModels

All three of these data structures can have relationships with tModels, but the meaning of the tModel differs in each context. tModels are an abstract concept representing standards, specifications, and documents. They're designed to be very general purpose so perhaps this is why they appear so confusing at first.

A tModel could be used to represent a taxonomy system such as the North American Industry Classification System (NAICS), which allows companies to be classified depending on the nature of their business activities.

Another use of a tModel is to represent a "service type" offered, such as a Stock Quote service. Table 1 shows the meaning of a tModel in different contexts.

What Is a Taxonomy?

A taxonomy is simply a classification system with a finite set of values. For instance, a geographic taxonomy such as ISO3166 allows

Author Bio

Andy Grove is a product manager at Cape Clear Software. He cofounded Orbware, a J2EE application server vendor, in 1999, and has extensive experience in building business systems using J2EE and XML technology.
ANDY.GROVE@CAPECLEAR.COM

WSJ ADVERTISER INDEX

classification of a business by its location.

Each taxonomy system is represented by a tModel and the classification of a particular entity within that taxonomy system is represented through use of the categoryBag structure, defined by UDDI. A categoryBag is a collection of keyedReference structures. Each keyedReference provides a name-value pair within the scope of the taxonomy system referred to by its tModelKey.

The XML fragment shown in Listing 1 demonstrates the use of the categoryBag and keyedReference structures within a businessEntity.

In this example, Cape Clear Software is categorized as "Software Publishers" within the NAICS taxonomy and "US" within the ISO3166 taxonomy. These taxonomies are referred to by their tModelKey references. When new tModels are registered, they're automatically given tModelKeys, which are guaranteed to be a Universally Unique Identifier (UUID).

tModel Categorization and Identification

In the previous section, we saw how business entities and business services can be identified and categorized according to taxonomies described by tModel structures. We've also seen how binding templates refer to a technical specification described in a tModel structure. We'll now look at how tModels themselves are identified and classified.

The UDDI specification provides some core tModels that must be supported in all UDDI repositories. These include standard industry classification and identification taxonomies, such as NAICS, UNSPC, and D-U-N-S. Another key tModel is one representing the "UDDI Types Taxonomy."

The purpose of the UDDI Types Taxonomy is to allow tModels to be classified according to their purpose. For instance, the NAICS taxonomy tModel is classified as a "categorization taxonomy" within the UDDI Types Taxonomy. Other valid values within the UDDI Types Taxonomy are "identifier" and "namespace." Table 2 shows the valid classifications within the UDDI Types Taxonomy.

TABLE 2: UDDI Types Taxonomy

ID	Description
tModel	These types are used for tModels
identifier	Unique identifier, e.g. D-U-N-S
namespace	Namespace
categorization	Categorization taxonomy e.g. NAICS
specification	Specification for a Web service
xmlSpec	Specification for an XML-based Web service
soapSpec	Specification for a SOAP-based Web service
wsdlSpec	Specification for a WSDL-based Web service
protocol	Protocol
transport	Wire/transport protocol
signatureComponent	Signature component

Registering a Web Service

Now that we've covered the main structures within a UDDI entry, we'll look at a real-world example of publishing a Web service within UDDI. The example is that of an Airport Weather service that can provide weather information for international airports. As a starting point, assume the Web service has already been developed and there's a valid WSDL document to describe the interface to the service.

The first step is to create a "Service Type" tModel to represent the Airport Weather service. This tModel is the generic specification of the Airport Weather service and this allows for the creation of actual Web services (business service entries) that are classified as being of type "Airport Weather." This is the equivalent of the separation of interface and implementation in object-oriented languages, such as Java. Any business entity may register business services referring to this service type.

Because this Web service is based on WSDL, this tModel should be classified within the UDDI Types Taxonomy (uddi-org:types) as being of type "wsdlSpec." This implies that the overviewDoc within the tModel contains a URL pointing to the WSDL document describing the interface to the Web service. Without this classification, there would be no standardized way for consumers of the service to know where to find a definition of the interface.

Listing 2 is the resulting tModel for the Airport Weather service. It's listed in the public UDDI operator cloud and can be queried from either the IBM or the Microsoft Web site.

The next step is to list a business service to implement this tModel. The business service uses the bindingTemplate structure to reference the Service Type tModel (see Listing 3).

Software applications can make use of the UDDI Types Taxonomy to limit searches to Web services that are based on WSDL, as shown in this sample UDDI request:

```
<find_tModel xmlns="urn:uddi-org:api"
generic="1.0" maxRows="1000">
  <categoryBag>
    <keyedReference
 keyName="Specification for a web serv-
ice described in WSDL"
 keyValue="wsdlSpec"
 tModelKey="uuid:c1acf26d-9672-4404-
9d70-39b756e62ab4"/>
  </categoryBag>
</find_tModel>
```

Conclusion

UDDI provides a technical infrastructure for publishing details of Web services on the Internet for use by other companies, or on the intranet for use by other depart xments. UDDI will become the key integration point for products supporting Web services. Runtime products will expose their capabilities through UDDI and developer tools will use UDDI to find those services.

There are still many business issues to be addressed around the use of UDDI on the Internet, such as how to ensure the validity or usefulness of data registered and how businesses will agree on contractual terms for the use of these services. It's unlikely that technology alone will be able to solve these problems but UDDI does provide a good framework for enabling both human and programmatic interaction between organizations. In the short term, the use of private UDDI repositories within intranets is likely to play a bigger role in the proliferation of the Web services paradigm.

References

- UDDI specifications: www.uddi.org
- CapeScience UDDI repository (Web interface): wwwlcapescience.com/uddi
- Cape Clear's public UDDI entry: http://uddi.microsoft.com/discovery?businessKey=5E4B4CDA-3448-4B88-ACE1-8584D43B7703

Listing 1 businessEntity categorization

```
<businessEntity
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       businessKey="5E4B4CDA-3448-4B88-ACE1-8584D43B7703"
       operator="Microsoft Corporation"
       authorizedName="Andy Grove"
       xmlns="urn:uddi-org:api">

  ...
  <name>Cape Clear Software</name>

  ...

  <categoryBag>
    <keyedReference
        tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
        keyName="Software Publishers"
        keyValue="5112" />
    <keyedReference
        tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88"
        keyName="United States"
        keyValue="US" />
  <name>Cape Clear Software</name>

  ...

  <categoryBag>
    <keyedReference
        tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
        keyName="Software Publishers"
        keyValue="5112" />
    <keyedReference
        tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88"
        keyName="United States"
        keyValue="US" />
  </categoryBag>

</businessEntity>
```

Listing 2 tModel representing Airport Weather

```
<tModel tModelKey = "uuid:7716711a-1231-483f-a4b9-36104341ba78"
         operator = "Microsoft Corporation"
         authorizedName = "Andy Grove">

  <name>Airport Weather</name>
  <description xml:lang="en">Checks weather at
   airports</description>
  <overviewDoc>
    <description xml:lang="en">WSDL description of
     Web Service</description>

<overviewURL>http://www.capescience.com/AirportWeather.wsdl<
 /overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey = "uuid:c1acf26d-9672-
      4404-9d70-39b756e62ab4"
                    keyName = "Specification for a web
                     service described in WSDL"
                    keyValue = "wsdlSpec" />
  </categoryBag>
</tModel>
```

Listing 3 businessService

```
<businessService serviceKey="55c48719-52dc-49d2-9a74-
 7c01383771b7"
                  businessKey="5e4b4cda-3448-4b88-
                   ace1-8584d43b7703">
      <name>Airport Weather Check</name>
      <description xml:lang="en">Airport weather
       information</description>

      <bindingTemplates>
        <bindingTemplate serviceKey="55c48719-52dc-49d2-
         9a74-7c01383771b7"
                         bindingKey="a318f871-bd15-4c5b-
                          955d-902468790f8c">
          <description xml:lang="en">SOAP endpoint for Web
           Service</description>
          <accessPoint
URLType="http">http://www.capescience.com/ccgw/GWXmlServlet<
/accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="uuid:7716711a-
             1231-483f-a4b9-36104341ba78">
              <description xml:lang="en">Web Service to
               check airport weather
              </description>
              <instanceDetails>
                <description xml:lang="en" />
                <overviewDoc>
                  <description xml:lang="en">WSDL for
                   Airport Weather
                  Service</description>

<overviewURL>http://www.capescience.com/AirportWeather.wsdl<
 /overviewURL>
                </overviewDoc>
                <instanceParms />
              </instanceDetails>
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
        <keyedReference tModelKey="uuid:c1acf26d-9672-4404-
         9d70-39b756e62ab4"
                        keyName="Specification for a web service
                         described in WSDL"
                        keyValue="wsdlSpec" />
      </categoryBag>
    </businessService>
```

WSJ News

WebGain and Flashline Offer Combined Solution

(Santa Clara, CA and Cleveland, OH) – WebGain, Inc., a provider of integrated software products to accelerate e-business development, and Flashline, a leading provider of software component reuse

solutions, have announced a strategic marketing and technology partnership to promote enterprise-wide, component-based development and software reuse. The companies have integrated Flashline Component Manager, Enterprise Edition (CMEE), with WebGain Application Composer.
www.webgain.com , www.flashline.com

JDJ Announces Record Circulation

(Montvale, NJ) – SYS-CON Media has announced a projected circulation of 104,254 copies for the June 2001 print edition of ***Java Developer's Journal***,, the world's leading *i*-tecnology magazine. This issue generated single-copy revenues of over $1 million. ***JDJ*** expects to announce a six-month target circulation of 97,325 (up from its previous six-month average of 89,652) for the period ending June 2001. In addition, the digital edition of its September 2001 issue delivered a circulation of 236,715.
www.javadevelopersjournal.com

Cape Clear Ships Rapid Application Development Tool

(Campbell, CA) – Cape Clear Software has announced the commercial release of its CapeStudio Web Services development product, a rapid application development tool that enables software developers to build and deploy Web services and to integrate software systems faster and easier than before. The new product includes support for Microsoft BizTalk schemas, a UDDI browser, improved XML mapping, and support for Enterprise JavaBeans general from WSDL.

An online demonstration of CapeStudio is available at www.capeclear.com/demo.

Flamenco Networks Launches Web Services Network

(Atlanta, GA) – Flamenco Networks has announced immediate availability of its Web services network, which reduces the time and costs required to implement and manage business-class Web services.

The Flamenco Network combines peer-to-peer network transport technology with a Web-based, centralized management site and allows customers to implement their Web services across any community, internal or external. The network complements existing Web services development tools and platforms as well as application integration solutions. Customers pay a set-up fee and a recurring monthly fee based on number of connections.
www.flamenconetworks.com

Killdara to Acquire Interbind

(Almonte, ONT) – Killdara Corp. has reached an agreement in principle to acquire Interbind, Inc. of New York. Interbind's interbindIBX, interbindSIO, and interbindCodeGen products will be integrated into Killdara's Vitiris Web Services product family.
www.killdara.com, www.interbind.com

ILOG and LogicLibrary Team to Support Web Services

(Mountain View, CA) – ILOG and LogicLibrary have announced that ILOG JRules will be used within LogicLibrary's new Logidex software solution. With ILOG JRules business rule management software, Logidex will radically simplify component-based development (CBD) and the implementation of Web services.

Logidex's approach is based on a Domain Reference Model that provides a framework for cataloging and accessing solution assembly assets based on the business processes, technical architectures, and customized business models they support. ILOG JRules 3.5 is the industry's first XML-centric business rule engine and the company takes an active role in the definition of business ruile standards.
www.logiclibrary.com, www.ILOG.com

Shinka Technologoes Ships Business Integration Platform 1.3

(San Francisco, CA) – Shinka Technologies has announced the availability of Business Integration Platform 1.3, which features built-in support for Lotus Notes. The new version allows customers to integrate existing Lotus Notes applications via Web services into other applications. Existing systems based on Web services can also be integrated into Lotus Notes applications.
www.shinkatech.com

What's Happening in Web Services?

WRITTEN BY

Steve Benfield

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

Steve can be contacted at steve@sys-con.com

So, what is going on in the world of Web services? I'm looking at a ton of analyst reports saying Web services won't be mainstream for another two or three years. One firm says that only 16% of companies will use Web services this year. I suppose it comes down to your definition of Web services. Which is another question I get a lot—what is a Web service? Some people have a very broad definition and include Web sites that execute functionality. Some have a narrow definition and only include a SOAP-accessible piece of functionality with WSDL published to a UDDI server. Some define it as XML over HTTP regardless of whether any standards are followed. So when someone says Web services will be adopted by 35% of companies within the next 87.3 days, what does that mean?

I always hear people say things like, "That is a Web Service with a capital W and capital S," referring to something that implements all of the standards. A Web service is one that doesn't – i.e., XML over HTTP. There's another group that uses the phrase *services-oriented architecture* to encompass both Web services and things like EJBs. Or, we can have multi-channel systems – distributed systems using XML over HTTP or EJBs or .NET going to multiple audiences such as browsers and wireless devices.

I'd like to make one suggestion: Why don't we call them WebServices – one word. Then at least we'd have a better chance at finding them on search engines and ignoring the consulting and network companies. A long shot, I know. But I've been wanting to say it for a while.

There's still confusion in the mass marketplace about what Web services are and what the benefit is. Luckily, I think a lot of the superhype has died down and we're talking about just plain normal hype. Here's an example of superhype – Web services let you build "self-healing" applications. Right. Nothing in software is self-healing – unless you specifically write healing code. And that doesn't sound self-healing, it sounds like someone wrote a lot of code to make something appear to be self-healing. Once something that wasn't part of the self-healing algorithm breaks, so much for self-healing.

Here's a case where self healing could work: You buy things from the Ubiquitous Widget Company, including Widget #1024. Today you ask them for widgets and their site is down or they are out of stock or they put you on credit hold (did I mention you were a dot-com?). Time for plan B: buy from another supplier. You go to the UDDI taxonomy and find widget companies. Then you find the order entry service. Then you get pricing and delivery terms. You then create an account with them. You place an order. Hope the quality is good.

OK, let's look at the assumptions:

- Widget suppliers are well defined in the public UDDI taxonomy to find supplier B.
- You can find the appropriate Web services you need for supplier B. How do you programmatically know which one is the order-entry service?
- Once you find the service, how do you map your field names to their field names?
- How does Supplier B know what widget #1024 is?

"In the end, humans are still involved in almost every step of the way with Web services."

These are some of the assumptions. Essentially, we say that businesses and products all follow a well-defined–and well-understood–taxonomy. That we all use the same names for our Web services–or at least have some sort of metadescription of common services. After that, our services use a common set of element or field names–that Customer ID is the Customer ID all around the world and not CustID or CID or ID or CustNum. Then we assume that the semantics are the same. Is Price list price, my price? Is there a discount field used to calculate price? There will be a lot of wailing and gnashing of teeth before we see the day all of this is true.

In the end, humans are still involved in almost every step of the way with Web services. Perhaps we can have a partially self-healing application. We can define the five suppliers we do business with, and the mappings from our data to their Web service definitions. And we can have tables that map our part numbers to theirs. A lot of people already have all of these. What it requires is the programmer to write the Web service that takes all of this into account.

What does mainstream really mean? Clearly, no one will be doing the self-healing, pie-in-the-sky kind of Web services soon. However, many companies are using XML over HTTP today. Are they using Web services? I think they are – it covers my intuitive definition and that of many analysts. I think they will quickly move to more formal Web services. The massive interoperability that results will bring huge savings when it comes to building systems and applications across an enterprise. That alone is worth the price of admission to Web services – even if it isn't self-healing.

Silverstream

www.

Cape Clear

www.